# LE PETIT LIVRE DU




TRADUIT PAR P.S.I.

# LE PETIT LIVRE DU ZX 81

version française ISBN : 2.86595.036.0
version originale ISBN : 0.9507302.2.X

Traduction de l'ouvrage anglais

**The ZX 81 Pocket Book**



Imprimé en France

# LE PETIT LIVRE DU ZX 81

par
TREVOR TOMS
traduction de Jean-Pierre Richard




1982

# SOMMAIRE

# INTRODUCTION

Le ZX81 a succédé très rapidement au ZX80, l'un des ordinateurs domestiques les plus vendus. Quelles ont été les causes déterminantes d'un tel succès ? Son prix très abordable d'une part et sa simplicité d'emploi d'autre part : une combinaison unique !

Le nouvel ordinateur ZX81, commercialisé par la firme Sinclair depuis mars 1981, présente un certain nombre d'améliorations par rapport à son prédécesseur : un langage Basic plus évolué, une meilleure finition et un prix plus avantageux.

Dans ces conditions, pourquoi ne rencontrerait-il pas le même succès que le ZX80 ? La plupart des revues spécialisées ont fourni des commentaires enthousiastes sur ses possibilités. Il convient particulièrement à tous les débutants désireux de s'initier à la programmation.

Le contenu de ce livre se présente très différemment de celui du "ZX80 Pocket Book" ; tout en essayant de conserver la qualité de celui-ci, nous avons adopté un style plus concret et une présentation moins rigoureuse (le fruit de l'expérience...).

Si vous êtes déjà possesseur d'un ZX80, procurez-vous sans délai l'extension mémoire ROM 8K avant de vous plonger dans la lecture de ce livre ! Tous les programmes sont écrits avec les instructions Basic du ZX81 et sont incompatibles avec la version ZX80 initiale : il ne s'agit pas d'un ouvrage sur le ZX80 dont la couverture a été rapidement rebaptisée "ZX81" pour l'occasion.

Cette précision est importante dans la mesure où les deux modèles sont très différents.

La majorité des ZX81 étant vendue sans l'extension mémoire 16K RAM, une bonne part des programmes présentés dans ce livre nécessite seulement une capacité mémoire de 1K.

Par contre, l'extension mémoire est sans nul doute indispensable pour tirer le meilleur parti de l'ordinateur. Nous vous conseillons vivement cet achat si le virus de la programmation vous atteint. Peut-être que si vous écriviez au Père Noël... ?
Les débutants trouveront sans aucun doute dans ce livre quelque chose de plus stimulant que des programmes de jeux à recopier et à exécuter sans création personnelle. Nous souhaitons que les explications fournies au fil des chapitres répondent à votre attente. Il se pourrait même que certains passages vous fassent sourire.

Si vous ne connaissez pas encore grand chose à la programmation, ne tombez pas dans le piège classique consistant à recopier les programmes conçus par quelqu'un d'autre : ce n'est pas de la programmation. La plupart des gens sont capables de taper à la machine !
Essayez au contraire de développer vos idées propres et n'hésitez pas à utiliser les idées fournies dans ce livre pour alimenter votre imagination. La satisfaction sera d'autant plus grande si vous réussissez à faire marcher un programme de votre "cru" et à le faire apprécier par quelques amis.

Nous avons également essayé de développer les "sous-programmes utilitaires" : ceux-ci ont le mérite de montrer comment exploiter certaines possibilités du ZX81 et en tirer le meilleur parti. Ces sous-programmes fournissent également un élément d'apprentissage intéressant pour les débutants, puisqu'ils peuvent venir s'insérer facilement dans d'autres programmes.

Un dernier point : il se peut que vous découvriez une erreur dans un programme. L'expérience prouve que la plupart des erreurs sont dues à des fautes de frappe et peuvent passer inaperçues assez longtemps. Quelle qu'en soit l'origine, vous les trouverez déplaisantes, surtout si vous avez passé du temps à entrer le programme en mémoire. Ne vous découragez pas. Il y a une explication logique pour tout : tout problème n'est en fait qu'une solution cachée !

# PRÉSENTATION

Avant tout, il convient de préciser certaines des conventions utilisées dans cet ouvrage :

***Listings de programmes***

1- Toutes les instructions clés sont soulignés.

2- Toutes les fonctions clés et les symboles (par exemple **<>** ou **>=** etc...) apparaissent en caractères gras.

3- Par souci de clarté, si un message à imprimer occupe plus d'une ligne, celui-ci n'a pas obligatoirement la même disposition sur le listing du programme et sur l'écran au moment où vous le tapez. Prenons l'exemple suivant qui est extrait du programme intitulé "COURSE DE SKI" :

```
9060 PRINT "  SELECTIONNEZ EN PREMIER LIEU",
           "LE NIVEAU DE L EPREUVE QUE",
           "VOUS DESIREZ TENTER."
```

Cette présentation "en colonne" correspond à l'affichage du message tel qu'il est obtenu par la commande RUN au moment de l'exécution du programme.

Pour entrer cette ligne en mémoire, il faut taper les trois messages à la suite les uns des autres sans oublier de les séparer par une virgule. Ainsi, après avoir appuyé sur la touche NEW LINE, vous verrez apparaître sur l'écran :

```
9060 PRINT " SELECTIONNEZ EN PRE
MIER LIEU","LE NIVEAU DE L EPREU
VE QUE","VOUS DESIREZ TENTER."
```

4- Dans le but d'améliorer la compréhension des méthodes de programmation employées, les listings sont agrémentés de commentaires ; ceux-ci sont situés entre parenthèses à droite de la page.

*Exemple :*

```
200 LET X=1                    (ceci est un commentaire).
```

***Les exemples***

Les explications fournies sont accompagnées d'un certain nombre de petits programmes ou sous-programmes à expérimenter. Certains sont conçus pour mettre en valeur un point particulier, principalement dans le chapitre "*OPTIMISATION DES PROGRAMMES*", où sont inclus de nombreux programmes d'évaluation des performances.

En les chronométrant, vous pouvez comparer l'efficacité des différentes méthodes de programmation ; les temps obtenus sont consignés dans un tableau à la fin du livre.

# SI QUEUE D'ANE M'ÉTAIT CONTÉE

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***



Commençons par une récréation : comment restituer une queue à cette pauvre bête qui a perdu la sienne ?

Le programme est exécuté sur le mode rapide (instruction FAST) : par conséquent, vous ne pouvez pas voir le dessin de l'âne de façon continue. Dès que vous appuyez sur une touche, tout est effacé de l'écran : vous devez alors deviner combien de temps il faut continuer à appuyer sur cette touche.

Quand vous relevez le doigt, la queue est décochée en direction de l'âne. Si vous avez réussi à l'atteindre, votre score est affiché sur l'écran. Par contre, si vous attendez trop longtemps, vous risquez d'occasionner quelques dommages supplémentaires à ce pauvre animal !

Le but consiste à "épingler la queue" en un minimum de coups.

Le dessin de l'âne programmé à la ligne 40 fait apparaître en gros caractères :

1,D
2,D
3,D
4,D

```
   1 REM ANE
   2 REM
  10 FAST
  20 LET T=10                              (initialisation de T
  30 LET D=INT (RND*T)+19                  (position de l'âne
  40 PRINT AT 1,D;"  ▘▘";AT 2,D;"▄█▀";AT 3,D;"▜▙";AT 4,D;"▛▛"
                                           (dessine l'âne
  50 LET N=1                               (compteur de tours
  60 LET X=NOT N                           (position de départ
 100 PAUSE 4E4                             (affiche
 110 POKE 16437,255
 120 IF INKEY$<>"" THEN GOSUB 1000         (touche appuyée ?
 130 IF C>=D THEN GOTO 300                 (dépasse l'âne ?
 140 IF X=D*2-1 THEN GOTO 200              (coup au but ?
 150 LET N=N+1                             (incrémentation du compteur N
 160 GOTO 100                              (encore un tour
 200 PRINT AT T,T;"PARFAIT";AT 14,0;N;" COUPS"  (gagné !
 210 GOTO 400
 300 IF X<=63 THEN PRINT AT T,T;"AIE   "        (perdu !
 400 INPUT Y$                              (attend un nouveau tour
 410 CLS
 420 RUN
1000 PLOT X,39                             (dessine la queue
1010 LET C=INT (X/2)                       (l'âne est-il touché ?
1020 IF C-D>=0 AND C-D<3 THEN UNPLOT X,39       (...si oui, efface
1030 IF INKEY$="" THEN RETURN              (retour au programme principal
                                            si pas de touche pressée
1040 LET X=X+1                             (position suivante
1050 IF X<64 THEN GOTO 1000                (encore sur l'écran ?
1060 PRINT AT T,T;" DOMMAGE "              (sortie de l'écran
1070 RETURN
```

# IDENTIFICATION D'UNE CHAINE DE CARACTÈRES

***Pourquoi identifier une chaîne de caractères ?***

Quelle utilité peut avoir un sous-programme permettant de rechercher une chaîne de caractères à l'intérieur d'une expression ? C'est ce que va nous montrer l'exemple suivant. Imaginez un programme comportant ces deux lignes :

```
100 PRINT "COMMENT SE NOMME LE BEBE LIEVRE?"
110 INPUT A$
```

Lancez l'exécution du programme. Que se passe-t-il ? Une fois la question posée, l'ordinateur attend une réponse. Celle-ci peut prendre plusieurs formes différentes et néanmoins exactes, telles que :

UN LEVRAUT

LEVRAUT

ON L APPELLE UN LEVRAUT

etc...

Si la réponse programmée est LEVRAUT, une simple comparaison entre A$ et la réponse exclut la première et la troisième éventualités !

C'est là où intervient le sous-programme : il permet de rechercher une chaîne de caractères à l'intérieur d'une expression de chaîne pour voir si elle est présente ou non.

Pour ceux qui connaissent le Basic Microsoft, ce sous-programme est comparable à l'instruction INSTR.

Deux variables sont nécessaires au fonctionnement de ce sous-programme :

**A$** : variable alphanumérique contenant l'expression de chaîne à analyser.

**W$** : variable alphanumérique contenant la chaîne de caractères à rechercher dans A$.

Après retour au programme principal, la variable X contiendra soit la valeur 0 (cas où A$ ne contient pas la chaîne W$) soit l'emplacement du premier caractère de la chaîne W$ à l'intérieur de A$.

Le sous-programme se présente de la façon suivante :

```
9200 REM X=INSTR(A$,W$)
9210 IF LEN W$=0 THEN GOTO 9290       (prévient l'erreur n° 3
9220 LET X=1                          (initialise le compteur de caractères
9230 LET Y=LEN W$                     (n° du dernier caractère
9240 IF Y>LEN A$ THEN GOTO 9290       (dépassement de chaîne ?
9250 IF W$=A$(X TO Y) THEN RETURN     (trouvé dans W$ ?
9260 LET X=X+1                        (sinon caractère suivant
9270 LET Y=Y+1                        (réactualise le n° du dernier caractère
9280 GOTO 9240                        (encore un essai
9290 LET X=0                          (met à zéro si pas trouvé
9295 RETURN                           (fin du sous-programme
```

Appliquons le sous-programme au programme suivant :

```
100 REM "JEU DES QUESTIONS ET DES REPONSES"
110 LET SCORE=0
115 LET TOTAL=0
120 LET Q$="COMMENT SE NOMME LE BEBE LIEVRE?"
130 LET W$="LEVRAUT"
140 GOSUB 8000
150 LET Q$="QUELLE EST LA CAPITALE DU PEROU?"
160 LET W$="LIMA"
170 GOSUB 8000
180 LET Q$="QUEL EST LE JOUR LE PLUS LONG?"
190 LET W$="21 JUIN"
200 GOSUB 8000
210 LET Q$="COMBIEN FONT 54.2 * 29.333?"
220 LET W$=STR$ (54.2*29.333)
230 GOSUB 8000
240 LET Q$="QUI EST LE PLUS BEAU ?"
250 LET W$="MOI"
260 GOSUB 8000
270 LET Q$="DATE DE LA BATAILLE DE MARIGNAN?"
280 LET W$="1515"
290 GOSUB 8000
300 LET Q$="QUI A INVENTE L ECOLE ?"
310 LET W$="CHARLEMAGNE"
320 GOSUB 8000
```

```
 330 LET Q$="QUEL EST VOTRE SCORE ?"
 340 LET W$=STR$ SCORE
 350 GOSUB 8000
 400 SCROLL
 410 SCROLL
 420 PRINT "VOUS AVEZ ";SCORE;" SUR";TOTAL
 430 GOTO 9999
8000 SCROLL
8005 SCROLL
8010 PRINT Q$
8020 INPUT A$
8025 LET TOTAL=TOTAL+1
8030 SCROLL
8040 PRINT A$
8050 GOSUB 9200
8055 SCROLL
8060 IF X THEN GOTO 8110
8070 PRINT "C EST FAUX"
8080 SCROLL
8090 PRINT "LA REPONSE EST ■■";W$;"■■"
8100 RETURN
8110 PRINT "■■";W$;"■■ EST EXACT"
8120 LET SCORE=SCORE+1
8130 RETURN
9999 STOP
```

La variable Q$ contient la question, W$ la réponse. Le sous-programme situé à partir de la ligne 8000 pose la question, reçoit la réponse et vérifie si celle-ci est juste.

Si la réponse contient le mot-clé (affecté à la variable W$, le score est augmenté d'un point.

L'absence de restriction dans les réponses à donner à l'ordinateur font le charme de ce programme. Vous tapez ce que vous voulez et le sous-programme fait le reste à votre place !

Quelques-unes de ces questions sont ardues, mais si vous lisez la totalité de ce livre, vous trouverez le moyen de fournir la réponse correcte à coup sûr... et cela sans même avoir eu besoin de lire la question !!

Comment est-ce possible ? La solution vous sera fournie à la fin du livre si vous ne l'avez pas découverte vous-même entre-temps...

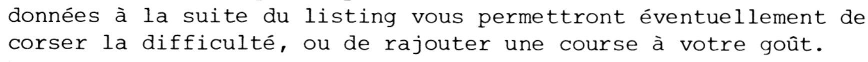
## COURSE DE SKI



Si vous avez toujours rêvé de slalomer à 120 km/h sans danger sur les pistes enneigées, voici la chance de votre vie !

Comme chacun sait, tout l'art du slalom consiste à passer du bon côté des "portes" en les évitant. Pour vous piloter entre elles, la touche "5" assure le dérapage à gauche, la touche "8" le dérapage à droite.

Le programme tel qu'il est présenté ici choisi au hasard une des deux courses. Quelques explications données à la suite du listing vous permettront éventuellement de corser la difficulté, ou de rajouter une course à votre goût.



```
   1 REM *** COURSE DE SKI ***
   2 REM
   3 REM
  10 GOTO 100
  20 REM CONTROLE LE MOUVEMENT
  30 LET D=CODE INKEY$              (code de la touche appuyée
  40 IF NOT D THEN RETURN           (touche incorrecte
  50 IF D=33 THEN LET S=S-1         (modifie la position du skieur
  60 IF D=36 THEN LET S=S+1
  70 LET S=S-INT (S/32)*32          (positionne le skieur sur
                                     l'écran
  80 RETURN
 100 REM INITIALISATION DU JEU
 110 GOSUB 9000                     (règles du jeu
 120 GOSUB 8000                     (choix de la course
 130 LET P=VAL C$(1 TO 2)           (départ de la course
 140 LET S=VAL C$(3 TO 4)           (position de départ du skieur
 150 LET H=0                        (initialise le compteur de
                                     fautes
1000 REM DEROULEMENT DE LA COURSE
1010 FOR X=5 TO LEN C$-1 STEP 2     (balaye la chaîne contenant
                                     la course
```

```
1020 SCROLL
1030 PRINT AT 21,P;"*";AT 21,P+G;"*";AT 21,S;"V"
1040 GOSUB 20                            (contrôle du mouvement
1050 LET P=P+VAL C$(X TO X+1)            (déplace les portes
1060 GOSUB 20                            (autre vérification
1070 IF S<=P OR S>=P+G THEN LET H=H+1    (porte bousculée ?
1080 NEXT X
2000 REM LIGNE D ARRIVEE
2010 SCROLL
2020 FOR X=P TO P+G                      (affiche la ligne d'arrivée
2030 PRINT AT 21,X;"-";
2040 NEXT X
2100 SCROLL
2110 SCROLL
2120 PRINT "VOUS AVEZ HEURTE";H;" PORTES"
2130 GOTO 9999
8000 REM SELECTION DE LA COURSE
8010 REM VOIR EXPLICATIONS JOINTES
8020 LET C$="141700000000-1-1-1-
     1-100000001010200-1-1-100010200-
     200-1-1-1000000000101010101012-
     2-20000000001010101010101010102020
     2000000000000-1-1-1010101-1-1-1-
     1-1-2-2-2-2-2-1-100000001010101O
     101-2-2-2-200000001010202020201O
     1010000000000000000"
8030 IF RND<0.5 THEN RETURN
8040 LET C$="0002000000000000000
     000000000101010101010101010100C
     0000002020202000O-1-1-1-1-1-1-1-
     1010101010000000000000000-1-1-1-
     1020202020101O1-2-2-2-2-2-2-2-1-
     1-1-1-10101-1-1000000000000"
8050 RETURN
9000 REM REGLES
9010 CLS
9020 SLOW
9030 PRINT TAB 8;"*** COURSE DE SKI ***"
9040 PRINT
```

```
9050 PRINT " VOUS DEVEZ PASSER ENTRE LES",
           "PORTES PENDANT LA DUREE DE LA",
           "DESCENTE. RENVERSER UNE PORTE",
           "OU SORTIR DE LA PISTE VOUS",
           "FAIT PERDRE DES POINTS."
9060 PRINT " SELECTIONNEZ EN PREMIER LIEU",
           "LE NIVEAU DE L EPREUVE QUE",
           "VOUS DESIREZ TENTER."
9070 PRINT "UTILISEZ LA TOUCHE""5""OU""8""",
           "POUR VIRER A GAUCHE OU A DROITE."
9080 PRINT
9100 PRINT "DANS QUELLE CATEGORIE ETES VOUS?"
           " (1) AMATEUR ",
           " (2) AMATEUR CHEVRONNE",
           " (3) PROFESSIONNEL",
9110 INPUT NIVEAU
9120 IF NIVEAU>0 AND NIVEAU<4 THEN GOTO 9200
9130 CLS
9140 PRINT "PARDON ?-";
9150 GOTO 9100
9200 LET G=3+(4-NIVEAU)*2                  (largeur de la porte
9210 PRINT "PRET?-"
9220 FOR X=3 TO 1 STEP -1                  (donne un délai de 3 s.
9230 PRINT X;"..";
9240 PAUSE 50
9250 NEXT X
9270 RETURN
```

La course est choisie au hasard dans le sous-programme démarrant à la ligne 8000 ; peut-être préférerez-vous la choisir vous-même, à moins que vous ne décidiez de remplacer, à votre idée, les deux courses existantes.

Les quelques explications qui suivent, vont vous indiquer comment créer vos propres descentes.

Après l'exécution du sous-programme (lignes 8000 à 8050), la variable alphanumérique C$ doit contenir un schéma de la descente du type :

**C$** (1 TO 2) première position du piquet gauche de la porte sur l'écran ; la position du piquet de droite est fonction de la catégorie choisie (celle-ci est entrée à la ligne 9110).

**C$**(3 TO 4) position de départ du skieur ; celui-ci est généralement placé deux "crans" à l'intérieur de la première porte. Si vous voulez corser la difficulté, rien ne vous empêche de faire démarrer le concurrent à l'extérieur de la porte.

**C$**(5 TO ) le reste de la chaîne alphanumérique est constitué d'une série de nombres à deux chiffres : ceux-ci, ajoutés à la position de la dernière porte, déterminent l'emplacement de la suivante. Les nombres inférieurs à -2 ou supérieurs à 02 sont exclus : en effet, le skieur ne peut se déplacer que de deux positions au maximum par "coup".

La description d'une course - pour le moins abrégée - pourrait se présenter de la façon suivante :

15 17 00 00 01 01 -1 -1 -2 -2 00 00

Que signifie en clair cet ensemble de chiffres ?

Le chiffre 15 représente la position du premier piquet (colonne 15), et le chiffre 17 la position de départ du skieur (colonne 17) ; aux deux premiers "coups", la position des portes reste la même (00). Les deux "coups" suivants déplacent la porte sur la droite (valeur positive) d'une seule position (01) à chaque fois.
Ensuite, la porte est déplacée sur la gauche d'un seul cran à la fois (-1), puis de deux crans à la fois (-2).
Les deux derniers "coups" laissent la position des portes inchangées (00).

Le programme affiche automatiquement la ligne d'arrivée et le nombre de portes "bousculées".

Le jeu progressant environ à la vitesse de 3 "coups" par seconde, la durée idéale d'une descente se situe entre 30 secondes et une minute. Bien sûr, rien ne vous empêche de concevoir une course de cinq minutes.

Faites attention à restreindre la course aux dimensions de l'écran ; dans le cas contraire, le code erreur B serait affiché. La course ne doit pas aller au-delà de la colonne 22, ou en deçà de la colonne 0. En effet, les colonnes supérieures à 22 ne sont pas utilisables, puisque la largeur des portes est de 9 crans en catégorie "amateur".

# OPTIMISATION DES PROGRAMMES

Dans ce chapitre, nous allons étudier les moyens dont dispose l'opérateur pour programmer efficacement.

Deux thèmes seront développés :
- l'optimisation de la mémoire vive (RAM).
- l'optimisation des temps d'exécution d'un programme.

Nous verrons que pour certains cas particuliers il est nécessaire d'opérer un compromis entre ces deux paramètres, le plus souvent au détriment du deuxième.

Ce dilemme est très fréquent sur les petits micro-ordinateurs ; même si le vôtre est équipé d'une extension 16K RAM, vous vous rendrez compte qu'une étude minutieuse est parfois nécessaire pour tout "caser" en mémoire (exemple : la version intégrale de STAR TREK).

Le manuel d'utilisation fourni par Sinclair détaille de façon tout à fait explicite le processus de stockage des variables et des programmes à l'intérieur du ZX81.

Néanmoins, certaines caractéristiques, qui ne sautent pas obligatoirement aux yeux, peuvent s'avérer très intéressantes en matière de programmation. C'est pourquoi, nous allons étudier le processus de stockage de chaque élément.

## REMPLACEMENT DES VALEURS NUMERIQUES

Si vous possédez un ZX81 de capacité mémoire 1K RAM (ce paragraphe a moins d'intérêt si vous êtes équipé d'une extension 16K), vous ne disposez lors de la mise en route que de 899 octets sur les 1024 prévus ; la différence (125 octets) est réservée au stockage des variables de système, ce qui représente un bon dixième de la capacité mémoire totale.

Le tampon mémoire écran nécessite au minimum 25 octets ; en effet, tout ce qui est entré dans le tampon d'affichage par l'intermédiaire des instructions PRINT ou PLOT consomme automatiquement de la place en mémoire. Cette remarque ne s'applique pas au modèle 16K RAM.

Par conséquent, vous disposez d'environ 850 octets à partager entre les lignes de programme, les variables et l'affichage (quand il est utilisé) ; l'interpréteur Basic nécessite lui aussi de la place en mémoire pour l'exécution du programme lui-même.

Dans une ligne de programme on compte :
- 2 octets par numéro de ligne (quel que soit le nombre de chiffres composant ce numéro)
- 2 octets pour indiquer la longueur de la ligne
- le nombre d'octets correspondant à la ligne elle-même (celui-ci se calcule aisément en comptant le nombre de touches pressées pour taper la ligne : voir explications ci-dessous)
- 6 octets supplémentaires pour chaque caractère numérique entré
- un octet supplémentaire pour passer à la ligne suivante.

Examinons en détail les exemples suivants :

| Ligne de programme | Emplacement mémoire |
|---|---|
| 10 LET X=Y | 9 octets au total : 2 pour le numéro de ligne, 2 pour la longueur de la ligne, 4 pour taper le contenu de la ligne lui-même (LET, X, =, Y), et un pour passer à la ligne suivante. |
| 300 LET X=1 | 15 octets au total : 2 pour le numéro de ligne, 2 pour la longueur, 4 pour taper le contenu, 6 pour le chiffre 1 et un pour passer à la ligne suivante. |
| 9000 DIM X(3,2) | 24 octets : 2 pour le numéro de ligne, 2 pour la longueur, 7 pour taper le contenu, 12 pour les chiffres "2" et "3" et un pour passer à la ligne suivante. |

Les exemples ci-dessus démontrent clairement que l'utilisation des expressions numériques dans un programme gaspille énormément de place en mémoire.

Pourquoi l'interpréteur Basic a-t-il été conçu de cette façon ? La raison principale est la suivante : la conversion des expressions numériques en valeurs à virgule flottante de 5 octets (et vice-versa) est un processus extrêmement lent. Peut-être l'avez-vous remarqué vous-même lors de l'affichage d'une expression numérique par l'intermédiaire de l'instruction PRINT.

Ceci explique pourquoi nous avions mentionné plus haut un dilemme entre occupation de la mémoire et temps d'exécution du programme.

Pour économiser lors de l'exécution du programme le temps nécessaire à la conversion, toutes les expressions sont stockées sous forme convertie.

Cette opération est implicite à chaque fois que vous entrez une ligne de programme en mémoire (ceci fait partie des routines de vérification de syntaxe).

La conclusion à tirer de tout ceci est la suivante : essayez de réduire au maximum l'utilisation des expressions numériques. Grâce à l'économie de place réalisée, vous pourrez entrer des programmes sensiblement plus importants.

Il existe plusieurs astuces permettant d'éviter l'utilisation d'expressions numériques ; certaines ralentissent le programme, d'autres font appel à une écriture particulière qui peut sembler originale à certains.

Une de ces astuces consiste à stocker dans une variable une valeur numérique couramment utilisée ; on emploie alors cette variable à la place de la valeur numérique elle-même.

*Exemple :*

```
10 IF X<>1 THEN LET A=1        (26 octets)
20 IF Y<>1 THEN LET B=1        (26 octets également)
```

... peut être avantageusement remplacé par :

```
 5 LET U=1                     (15 octets)
10 IF X<>U THEN LET A=U        (14 octets)
20 IF Y<>U THEN LET B=U        (14 octets également)
```

La première version occupe 52 octets alors que la deuxième n'en occupe que 43 ; on réalise ainsi une économie de 9 octets malgré la présence d'une troisième ligne de programme !

Malheureusement, ceci n'est pas strictement exact dans la mesure où le deuxième programme utilise une variable U, inexistante dans le premier. La variable numérique U à elle seule occupe 6 octets. L'économie réelle n'est donc que de 3 octets ; ce n'est pas le bout du monde, mais, à chaque fois que le chiffre "1" sera nécessaire dans le même programme, l'utilisation de la variable U à sa place économisera 5 octets.

Cette méthode présente également l'avantage de ne pas ralentir l'exécution du programme de façon sensible.

A titre d'exercice, comparez les temps d'exécution respectifs des deux programmes de référence suivants :

PR1A

```
  10 LET Y=0
  20 FOR T=1 TO 1000
  30 LET X=27
  40 NEXT T
9999 STOP
```

PR1B remplacer la ligne 30 par 30 LET X=Y

Pour ceux qui ne désirent pas chronométrer ces deux programmes (et les suivants), les temps d'exécution sont consignés dans un tableau à la fin du livre.

Lancez l'exécution du premier programme (PR1A) et chronométrez-le jusqu'à l'apparition du message 9/9999 (ne vous inquiétez pas si cela prend une ou deux minutes).

A présent, lancez l'exécution du deuxième programme (PR1B) sans oublier de modifier la ligne 30 comme indiqué. Chronométrez-le de la même façon. La différence de temps obtenue est due à la recherche du contenu de la variable Y et à l'affectation de celui-ci à la variable X ; cette opération n'existe pas dans le premier cas, puisque l'expression numérique est directement affectée à la variable X (ligne 30). Le programme effectuant 1000 itérations, l'erreur initiale obtenue sur une seule boucle est multipliée par 1000.

Une comparaison entre les deux résultats obtenus montre que la différence est négligeable.

Il n'est pas nécessaire d'affecter les valeurs 0 ou 1 à des variables déterminées ; en effet, ces deux valeurs peuvent être obtenues en utilisant une variable déjà existante dans le programme.

Supposons par exemple un programme utilisant une variable D. L'expression :

D=D donnera la valeur 1 (un nombre est obligatoirement égal à lui-même : le résultat de l'opération de comparaison est le chiffre 1).

**NOT** D=D donnera la valeur 0 (un nombre ne pouvant être différent de lui-même, le résultat de l'opération de comparaison est 0).

Ces deux expressions occupent moins de place en mémoire que les chiffres 0 ou 1 dans l'écriture d'une ligne de programme. Par contre, elles ont l'inconvénient de ralentir légèrement l'exécution. Si votre programme nécessite une exécution rapide, il est préférable

d'utiliser directement les expressions numériques.

Comparons les différents programmes de référence suivants :

PR2A

```
10 LET Y=20
20 FOR T=1 TO 1000
30 LET X=1
40 NEXT T
9999 STOP
```

PR2B remplacer la ligne 30 par 30 LET X=Y=Y

PR3A remplacer la ligne 30 par 30 LET X=0

PR3B remplacer la ligne 30 par 30 LET X=**NOT** Y=Y

Chronométrez les temps d'exécution de la même façon que précédemment et comparez les résultats obtenus pour les versions A et B de chaque programme. Ces résultats parlent d'eux-mêmes ; la version B occupe moins de place en mémoire. Quelle différence cela fait-il sur le plan de la vitesse d'exécution ?

Vous possédez maintenant tous les éléments nécessaires pour choisir vous-même en fonction de chaque cas particulier ce qui est le plus adapté : rapidité d'exécution ou économie de mémoire.

La ligne 5 LET U=1 nécessite 15 octets pour le stockage et 6 octets supplémentaires pour l'affectation de la variable U. Six octets sont économisés à chaque fois que la variable U est utilisée à la place du chiffre "1". Un rapide calcul montre que la variable U doit être utilisée au moins quatre fois dans le programme pour être "rentabilisée". On économise alors 4*6 = 24 octets, alors que la ligne 5 ne consomme que 21 octets.

Comme vous le voyez, il est préférable de réfléchir deux fois avant de gaspiller les octets sous prétexte d'en économiser !

Pour vous fixer les idées à ce sujet, reportez-vous à l'étude du programme "Jeu de Dés" ; celui-ci a été rédigé avec l'idée d'occuper le moins de place possible en mémoire.

## COMMENT EVITER L'UTILISATION DES VALEURS NUMERIQUES

Rares sont les programmes ne faisant pas appel à des valeurs numériques. Par contre, ils ne réclament pas obligatoirement la conversion en valeur numérique flottante sur 5 octets !

Sur le ZX81 1K RAM, ces variables à cinq octets peuvent occuper une place considérable en mémoire, particulièrement si les données se trouvent directement dans la ligne de programme (voir page 23).

Voici un exemple-type de programme nécessitant l'affectation préalable de plusieurs variables :

```
 10 DIM V(10)
 20 LET V(1)=22
 30 LET V(2)=9
 40 LET V(3)=11
 50 LET V(4)=17
 ....
 ....
110 LET V(10)=3
```

Le paragraphe précédent nous a montré que chaque ligne de ce type coûte au moins 24 octets, un luxe difficile à s'offrir dans le cas présent !

Une méthode pas très élégante consisterait à entrer ces données en mode direct et à sauvegarder le programme avec la variable indicée V contenant les valeurs pré-affectées.

Une meilleure solution consiste à mettre toutes les valeurs numériques dans une chaîne, puis à utiliser la fonction **VAL** pour les convertir. La seule contrainte, dans notre cas, est d'avoir deux chiffres par valeur même si celle-ci est inférieure à dix (3 devient 03).

Prenons l'exemple suivant :

```
10 DIM V(10)
20 LET V$="22091117036651042003"        (31 octets
30 FOR V=0 TO 9                         (23 octets
40 LET V(V+1)=VAL V$(V*2+1 TO V*2+2)    (57 octets
50 NEXT V                               (7  octets
```

L'économie réalisée dans ce cas (100 octets) est suffisante pour programmer éventuellement quelques lignes supplémentaires.

Le programme "Course de Ski" montre un exemple d'application de ce sous-programme.

Si la chaîne est particulièrement longue (dans l'exemple précédent, chaîne contenue dans V$), une fois la routine exécutée, vous pouvez libérer la place occupée par les variables en affectant une chaîne "vide" à la variable alphanumérique.

Ceci revient à rajouter une ligne au programme précédent :

```
60 LET V$=""
```

Là encore, chaque programme est un cas particulier. Il est quasiment impossible de créer un sous-programme universel qui puisse s'adapter efficacement à tous les cas. Nous avons simplement essayé de montrer dans ce paragraphe où et comment économiser de la place en mémoire. Il est possible que vous utilisiez des lignes de programmes très différentes de celles présentées ci-dessus. C'est pourquoi, nous présentons plus loin, pour ceux qui le désireraient, une routine relativement passe-partout.

Peut-être vous demandez-vous si l'utilisation d'une chaîne apporte réellement un gain de temps par rapport à l'utilisation d'une expression numérique. Pour vous en convaincre, essayez les deux programmes de référence suivants :

PR4A

```
  10 LET X$="99"
  20 FOR T=1 TO 1000
  30 LET X=1
  40 NEXT T
9999 STOP
```

PR4B remplacer la ligne 30 par `30 LET X=VAL X$`

Remarquez que le temps d'exécution du programme PR4A est sensiblement le même que celui des programmes PR1A, PR2A, et PR3A.

## ***STOCKAGE DES CHAINES DE DONNEES***

Certaines versions du Basic utilisent les commandes DATA et READ pour stocker des nombres ou des chaînes de caractères à l'intérieur d'un programme.

Bien que l'on puisse se passer de ces deux commandes, il serait parfois intéressant de mémoriser dans un programme une chaîne alphanumérique sans avoir recours aux tableaux de valeurs (ceux-ci occupent une place importante en mémoire).

Le stockage d'un ensemble de mots de longueurs différentes nécessite la constitution d'un tableau dont la longueur est celle du mot le plus long.

Ce type de stockage des données occupe beaucoup de place en mémoire ; en contrepartie, le temps d'exécution est très rapide puisqu'il suffit d'indiquer l'indice de la variable pour obtenir le mot complet.

Pour certains programmes la place disponible en mémoire est primordiale par rapport à la vitesse d'exécution. Dans ce cas, il est possible d'y adapter le sous-programme présenté ci-dessous (ou un

sous-programme équivalent).

Celui-ci permet d'extraire un mot ou une phrase d'une variable alphanumérique.

Un exemple d'application de cette routine est fourni avec le programme "Création d'un fichier d'individus".

Pour stocker des valeurs numériques, il vous suffit d'utiliser la fonction **VAL** avec la chaîne de caractères obtenue après l'exécution du sous-programme.

Le fonctionnement du sous-programme nécessite l'emploi des variables suivantes :

**D$** variable alphanumérique contenant la chaîne de caractères. Chaque mot est séparé par un caractère "/". Il est possible de remplacer ce dernier en modifiant les lignes 9030 et 9050 en conséquence.

**N** variable numérique contenant le numéro d'ordre du mot à extraire dans la chaîne D$.

**W$** variable alphanumérique recevant le mot extrait ; si le numéro d'ordre N du mot n'existe pas dans D$, W$ reste "vide". Pour extraire une valeur numérique par cette méthode, appliquer la fonction **VAL** à cette variable.

```
9000 REM MET LE N-IEME MOT DE D$ DANS W$
9005 LET XW=0
9010 LET W$=""
9015 FOR X=1 TO N-1
9020 LET XW=XW+1
9025 IF XW>LEN D$ THEN RETURN
9030 IF D$(XW)<>"/" THEN GOTO 9020
9035 NEXT X
9040 LET XW=XW+1
9045 IF XW>LEN D$ THEN RETURN
9050 IF D$(XW)="/" THEN RETURN
9055 LET W$=W$+D$(XW)
9060 GOTO 9040
```

D'aucuns penseront probablement que ce sous-programme n'économise pas beaucoup de place en mémoire. Manifestement, celui-ci n'a d'intérêt que s'il y a un grand nombre de données à stocker.

Les lignes contenant l'instruction REM peuvent être éventuellement omises lors de l'insertion de ce sous-programme dans un programme principal.

Ce sous-programme occupe en tout 222 octets (en comptant toutes les variables utilisées, mais en omettant l'instruction REM). Sans nul doute, l'extension 16K RAM est souhaitable pour exploiter pleinement cette méthode.

Malgré cela, si vous ne possédez que le modèle 1K, il vous reste encore quelques 500 octets disponibles. Aussi ne bannissez pas complètement cette routine de vos programmes !

Ne perdez pas de vue qu'une variable alphanumérique peut être affectée en mode direct avant l'exécution du programme ; elle peut être ensuite sauvegardée avec celui-ci sur cassette. Ce procédé évite l'utilisation encombrante de l'instruction LET.

## COMMENT STRUCTURER UN PROGRAMME

Le microordinateur ZX81 peut sembler rapide à celui ou à celle qui connaît peu ou pas du tout ce type de matériel.

Il faut cependant admettre, sans rien retirer à ses qualités (surtout son prix), qu'il est en réalité assez lent. Vis-à-vis de microordinateurs professionnels et même de certains microordinateurs individuels, le ZX81 ne soutient pas la comparaison. Ceci a une influence importante sur la vitesse de réponse de l'ordinateur en mode conversationnel (celui-ci nécessitant l'entrée des données au clavier).

Au cours des paragraphes précédents, nous avons vu comment optimiser l'utilisation de la mémoire dans un programme ; maintenant, abordons les problèmes inhérents à la vitesse d'exécution.

Dans un premier temps, nous allons considérer le cas d'une utilisation fréquente d'un ou plusieurs sous-programmes. Quand le ZX81 rencontre au sein d'un programme les instructions GOTO ou GOSUB, il passe en revue toutes les lignes de ce programme (en repartant de la première) jusqu'à ce qu'il trouve l'adresse demandée.

Par conséquent, si le programme doit être exécuté le plus rapidement possible, *placez de préférence les sous-programmes souvent appelés dès les premières lignes* (voir le programme "Course de Ski", page 16).

Si vous observez cette règle, vous obtiendrez un programme du type :

```
   1 REM nom du programme etc...
  10 GOTO 1000
  20 REM sous-programme très utilisé
  ....
  ....
1000 REM programme principal
  ....
  ....
```

```
8999 STOP

9000 REM sous-programme peu utilisé
        (initialisation etc...)
```

Les programmes de référence suivants montrent comment la vitesse d'exécution d'un programme peut varier en fonction de son mode d'écriture :

```
PR5A        10 LET X=1
            20 FOR T=1 TO 1000
            30 IF X=1 THEN NEXT T
            40 NEXT
          9999 STOP
```

PR5B remplacer la ligne 30 par 30 IF X=0 THEN NEXT T

PR6A remplacer la ligne 30 par 30 IF X THEN NEXT T

PR6B remplacer la ligne 30 par 30 IF NOT X THEN NEXT '

Le résultat de ces tests montrent que l'exécution d'une opération de comparaison est plus rapide si le résultat est 0 (comparaison "fausse"). Bien évidemment, si des lignes de programme supplémentaires sont indispensables pour s'assurer que la comparaison est fausse, le bilan est plutôt négatif ! Par contre, si le choix est possible, arrangez-vous pour que dans la majorité des cas, les opérations de comparaison soient "fausses".

On peut tirer un autre enseignement des deux derniers programmes de référence : que ce soit pour économiser temps ou mémoire, il est préférable de tester une valeur (vraie ou fausse) en mode direct sans utiliser les expressions du type X=1 ou X=0 (voir PR6A et PR6B).

Ceci est uniquement valable si la valeur contenue dans la variable est égale ou différente de zéro (en effet, la condition "vraie" donne un résultat différent de zéro).

Etudiez attentivement ces résultats : incontestablement, vos programmes pourront bénéficier de ce type de renseignement.

D'autre part, un certain nombre de programmes utilise une variable qui est incrémentée de une unité à chaque itération de boucle. A la place, il est possibile d'utiliser les instructions FOR/NEXT. Est-ce bien rentable ? C'est ce que vous pourrez constater en chronométrant les deux programmes de référence suivants :

```
PR7A        10 LET T=1
            20 IF T=1000 THEN STOP
            30 LET T=T+1
            40 GOTO 20
```

PR7B

```
  10 FOR T=1 TO 1000
  20 NEXT T
9999 STOP
```

Nous vous laissons apprécier vous-même quelle est la version la plus intéressante.

## EXPRESSIONS NUMERIQUES

L'utilisation des expressions numériques rend souvent l'écriture d'un programme plus compacte. Qu'elles permettent (ou non) d'accélérer l'exécution d'un programme, elles rendent fatalement celui-ci difficile à comprendre. Par contre, utilisées à bon escient, elles peuvent réduire le temps de travail en mémoire.

Nous avons vu que le ZX81 exprime le résultat de toute opération de comparaison par la valeur 0 (opération fausse) ou la valeur 1 (opération vraie).

Cette caractéristique est exploitée couramment lors de l'affectation à une variable d'une valeur elle-même fonction d'une ou plusieurs autres variables. Un exemple d'application est fourni dans le programme "l'ardoise magique" aux lignes 110 et 120 :

```
110 LET DX=(D=5)*-1+(D=8)
120 LET DY=(D=6)*-1+(D=7)
```

En fonction de la valeur prise par la variable D, on obtient les quatre combinaisons suivantes pour DX et DY :

| Valeur de D | Contenu de DX | Contenu de DY |
|---|---|---|
| 5 | -1 | 0 |
| 6 | 0 | -1 |
| 7 | 0 | +1 |
| 8 | +1 | 0 |

Une autre écriture possible est la suivante :

```
110 LET DX=0
114 LET DY=0
118 IF D=5 THEN LET DX=-1
122 IF D=6 THEN LET DY=-1
126 IF D=7 THEN LET DY=1
128 IF D=8 THEN LET DX=1
```

... cette dernière solution occupe évidemment beaucoup de place en mémoire (142 octets contre 82 pour la version initiale !).

La variable D ne pouvant prendre qu'une seule valeur à la fois, la comparaison "(D=5)" est soit vraie (valeur 1), soit fausse (valeur 0). Le produit obtenu en multipliant le résultat par un nombre n est égal à n si la comparaison est vraie ou à 0 si la comparaison est fausse.

Cette astuce peut être utilisée dans le cadre des économies de place en mémoire. Imaginons un programme permettant de tester une variable :

```
IF A$(1)="Y" THEN...
```

Si cette expression doit être utilisée plusieurs fois au cours du programme, il est plus économique d'écrire :

```
LET A=(A$(1)="Y")
```

... puis de tester si la variable A contient la valeur 1 ou 0

```
IF A THEN
```

Il est évident que dans ce cas la variable A reflète le résultat du test au moment de l'affectation de la variable. Si le contenu de A$ est modifié, la variable A ne donne plus obligatoirement le résultat correct, à moins d'opérer une ré-affectation.

Par conséquent, cette méthode peut être utilisée dans un sous-programme gérant une entrée du type "répondre par OUI ou NON" :

```
8000 REM GESTION D UNE REPONSE OUI/NON
8010 PRINT "REPONDRE PAR OUI OU NON"
8020 INPUT Y$
8030 LET Y=CHR$ CODE Y$="O"
8040 RETURN
```

Ce sous-programme est suffisamment souple dans le contrôle de la réponse entrée à la ligne 8020 pour être adapté selon votre goût. L'essentiel est que le contenu de la variable Y reflètant la réponse donnée (OUI ou NON) soit testée de la façon suivante :

IF Y THEN PRINT "VOUS AVEZ REPONDU OUI" (ou ce que vous voulez)

ou

IF NOT Y THEN PRINT "VOUS AVEZ TAPE NON"

Ce sous-programme "coûte cher" en octets et ne trouve pas vraiment de justification s'il n'est appelé qu'une ou deux fois par le programme principal. Par contre, utilisé plusieurs fois il peut s'avérer rentable en termes d'occupation de mémoire, puisque la totalité de l'expression a déjà été évaluée (comparer les résultats obtenus pour les programmes de référence PR5 et PR6).

***Note aux utilisateurs de ZX80***

Avec la ROM 4K d'origine, les opérateurs AND et OR utilisaient l'algèbre de BOOLE (les nombres entiers n'occupant que deux octets). Les mêmes astuces ne peuvent être employées avec la ROM 8K : en effet, les opérateurs AND et OR travaillent dans ce cas sur cinq octets et donne une valeur de 1 pour "vrai" (au lieu de -1). Remarquez également que la valeur -1 n'indique plus que tous les octets sont à l'état 1.

## L'ETOILE FILANTE

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***

Une étoile passe, essayez de l'arrêter en appuyant sur une des touches numériques (1 à 9). Il est facile de tout arrêter en utilisant la touche "9", mais l'intérêt du jeu consiste à obtenir le score le plus bas possible en se servant des touches correspondant à la valeur la plus basse possible ("3" ou "2"). Plus le score est bas, meilleur il est. Si vous ne réussissez pas avec une touche, essayez avec une touche correspondant à un chiffre supérieur. Ce jeu est difficilement réalisable sur un ZX80.

```
 10 PRINT AT 10,0;"*"
 20 FOR X=1 TO RND*50+25
 30 NEXT X
 40 FOR X=1 TO 10
 50 PRINT AT 10,(X-1)*3;" ";AT 10,X*3;"*"
 60 LET Z=CODE INKEY$
 70 IF Z AND Z-28>=X AND Z-28<10 THEN GOTO 110
 80 NEXT X
 90 PRINT AT 14,10;"RATE"
100 GOTO 120
110 PRINT AT 14,10;"SCORE:";Z-28-X
120 PAUSE 4E4
130 IF INKEY$<>"" THEN GOTO 130
140 CLS
150 RUN
```

## PARTIE DE BASKET-BALL

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***



Voici un nouveau programme qui va sans nul doute vous inciter à jouer... et à gagner ! Dans le cas présent, votre rôle consiste à mettre le ballon au panier. Le ballon apparaît en haut et à gauche de l'écran puis, quelques secondes après, amorce sa chute. Celle-ci est régie par un programme simulant la force de gravité.

Appuyez sur la touche "8" pour propulser le ballon vers la droite ; plus vous appuyez sur la touche "8" et plus la force appliquée au ballon est élevée. Rien ne vous empêche d'agir par tâtonnements en redonnant un "petit coup" sur la touche "8" de temps en temps.

A chaque partie, la position du panier est modifiée, ce qui évite une maîtrise trop rapide du jeu une fois que l'on a "le compas dans l'oeil".

Pour gagner, il suffit que le ballon heurte le panier. C'est ce que vous n'allez pas tarder à découvrir en disputant votre première partie !

La simulation de la force de gravité est créée entre les lignes 200 et 260 du programme. D'autre part, le temps de chute est compté dans la variable Y. Pendant un temps t (temps écoulé depuis le point de départ), le ballon parcourt une distance égale à $32t^2$. Dans ce programme, un facteur 10 est appliqué à la variable Y pour que le jeu se déroule sur toute la surface de l'écran.

```
 5 CLS
10 LET P=INT (RND*26)+5          (positionne le panier
20 PRINT AT 21,P-1;"(-)"         (dessine le panier
30 LET P=P*2                     (définit la zone "touchable"
40 LET X=0                       (position horizontale
50 FOR Z=1 TO RND*50+50          (pause
```

```
 60 UNPLOT 0,39                         (fait disparaître et
 70 PLOT 0,39                           (réapparaître le ballon
 80 NEXT Z
 90 LET N=0                             (élan du ballon
200 FOR Y=0 TO 11                       (temps de chute
210 IF INKEY$="8" THEN LET N=N+1        (force modifiée ?
220 LET X=X+N                           (réactualise la force
230 IF X>63 THEN GOTO 400               (sorti de l'écran ?
240 PLOT X,39-32*(Y/10)**2              (dessine nouvelle position du ballon
250 IF INKEY$="8" THEN LET N=N+1        (force de nouveau modifiée ?
260 NEXT Y
300 IF ABS (P-X)<4 THEN PRINT AT 11,15;"**HOURRAH **" (gagné !
310 PAUSE 150                           ("désensibilise" le clavier
320 IF INKEY$="" THEN GOTO 320          (touche appuyée ?
330 RUN                                 (nouvelle partie
400 PRINT AT 18,6;"RATE"                (sorti de l'écran
410 GOTO 310
```

Si vous possédez un ZX80 équipé de la ROM 8K, modifiez le programme précédent de la façon suivante :

Remplacez les lignes 50 à 80 par :

```
50 PAUSE 4E4
60 POKE 16437,255
```

Ajoutez les lignes suivantes :

```
255 PAUSE 25
256 POKE 16437,255
```

Remplacez les lignes 310 et 320 par :

```
310 PAUSE 4E4
320 POKE 16347,255
```

## PARTIE DE GOLF

***CAPACITE MEMOIRE NECESSAIRE : 16K RAM***



Vous avez votre "club" bien en main ? Alors concentrez-vous sur le trou (symbolisé par un 0 en blanc sur fond noir) et ne ratez surtout pas la balle (X en blanc sur fond noir). La direction et la force du jet imprimé à la balle sont entrées aux lignes 330 et 390.

Ce programme utilise une routine permettant le traçage d'une ligne (se reporter au manuel Sinclair ZX81). Celle-ci a été adaptée pour les besoins de la cause ; elle permet aussi bien de tracer que d'effacer la ligne.

```
  1 REM *** PARTIE DE GOLF ***
 10 GOSUB 4000                      (donne règles du jeu
 20 CLS
 30 REM DESSIN DES LIMITES DU TERRAIN
 40 FAST
 50 GOSUB 2000
100 LET TROUX=INT (RND*30)+1        (repère le trou
110 LET TROUY=INT (RND*18)+1
120 LET BALLEX=INT (RND*30)+1       (place la balle
130 LET BALLEY=INT (RND*18)+1
140 IF BALLEX=TROUX AND BALLEY=TROUY THEN GOTO 120
150 LET COURS=0
190 SLOW
200 PRINT AT TROUY,TROUX;"O"        (utilise un "O" inverse
210 PRINT AT BALLEY,BALLEX;"X"      (utilise un "X" inverse
300 REM BOUCLE PRINCIPALE
310 GOSUB 3100                      (efface la ligne 20
320 PRINT "DIRECTION?"
```

```
330 INPUT XD
340 GOSUB 3000                          (efface la question
350 IF XD<0 OR XD>12 THEN GOTO 300      (vérifie la direction
360 PRINT TAB 22;XD
370 GOSUB 3000
380 PRINT "FORCE"
390 INPUT XS
400 LET XS=INT XS                       (valeur entière de la force
410 GOSUB 3000                          (efface la question
420 PRINT TAB 28;XS
500 LET A=BALLEX*2                      (prépare la trajectoire
510 LET B=INT ((BALLEY*(-2))+42)
520 LET XD=(XD*(-1))+15                 (calcule la bonne direction
530 IF XD=12 THEN LET XD=0
540 LET C=A+INT (COS ((PI/6)*XD)*XS)    (destination
550 LET D=B+INT (SIN ((PI/6)*XD)*XS)    (destination
560 LET XA=A
570 LET XB=B
580 GOSUB 1500                          (vérifie la nouvelle position
590 LET P=1                             (commande le traçage de la
                                                              balle
600 GOSUB 1000                          (trace la ligne
610 PRINT AT BALLEY,BALLEX;" ";         (efface l'ancienne balle
620 LET BALLEY=INT ((D-42)/(-2)+0.5)    (calcule position de la nouvelle
630 LET BALLEX=INT ((C+0.5)/2)                                balle
640 LET COUPS=COUPS+1
700 IF BALLEX=TROUX AND BALLEY=TROUY THEN GOTO 800
                                        (balle dans le trou ?
710 LET A=XA                            (efface la ligne
720 LET B=XB
730 LET P=0                             (commande l'effacement
740 GOSUB 1000                          (efface la ligne
750 GOTO 200
800 GOSUB 3400                          (affiche "PLOP"
810 PRINT AT 20,0;"REUSSI"              (message de victoire
820 PRINT "EN ";COUPS;" COUPS";
830 GOSUB 3500                          (affiche appréciations
840 IF INKEY$ ="" THEN GOTO 840         (attend une nouvelle partie
850 CLEAR
860 GOTO 20
```

(voir le manuel Sinclair

```
1000 LET U=C-A
1010 LET V=D-B
1020 LET D1X=SGN U
1030 LET D1Y=SGN V
1040 LET D2X=SGN U
1050 LET D2Y=0
1060 LET M=ABS U
1070 LET N=ABS V
1080 IF M>N THEN GOTO 1130
1090 LET D2X=0
1100 LET D2Y=SGN V
1110 LET M=ABS V
1120 LET N=ABS U
1140 LET S=INT (M/2)
1150 FOR I=0 TO M
1160 IF P THEN PLOT A,B
1165 IF NOT P THEN UNPLOT A,B
1170 LET S=S+N
1180 IF S<M THEN GOTO 1230
1190 LET S=S-M
1200 LET A=A+D1X
1210 LET B=B+D1Y
1220 NEXT I
1230 LET A=A+D2X
1240 LET B=B+D2Y
1250 NEXT I
1260 RETURN
1500 REM VERIFIE NOUVELLE POSITION DE LA BALLE
1510 LET ERREUR=(C<2 OR C>62 OR D<6 OR D>42)
1520 IF NOT ERREUR THEN RETURN
1530 PRINT AT 21,0;"DANS L HERBE-PENALITE"
1540 LET COUPS=COUPS+1
1550 IF C<2 THEN LET C=2
1560 IF C>62 THEN LET C=62
1570 IF D<6 THEN LET D=6
1580 IF D>42 THEN LET D=42
1590 RETURN
2000 REM DESSINE LIMITES DU TERRAIN
2010 LET Y=43
```

```
2020 GOSUB 2100
2030 LET Y=4
2040 GOSUB 2100
2050 LET X=0
2060 GOSUB 2200
2070 LET X=63
2080 GOSUB 2200
2090 RETURN
2100 REM DESSINE LIGNE HORIZONTALE
2110 FOR X=0 TO 63
2120 PLOT X,Y
2130 NEXT X
2140 RETURN
2200 REM DESSINE LIGNE VERTICALE
2210 FOR Y=4 TO 43
2220 PLOT X,Y
2230 NEXT Y
2240 RETURN
3000 REM EFFACE LA QUESTION
3010 PRINT AT 20,0;"   <-20 espaces->   "
3020 PRINT AT 20,0;
3030 RETURN
3100 REM EFFACE LIGNE 20
3110 PRINT AT 20,0;"   <-63 espaces->   "
3120 PRINT AT 20,0;
3130 RETURN
3400 REM PLOP
3410 IF TROUX<2 THEN LET TROUX=2
3420 IF TROUX>29 THEN LET TROUX=29
3430 PRINT AT TROUY,TROUX-2;"PLOP"
3440 RETURN
3500 REM APPRECIATIONS
3510 IF COUPS=1 THEN PRINT "-EN UN COUP";
3520 IF COUPS=2 THEN PRINT "-PAS MAL...";
3530 IF COUPS<6 THEN GOTO 3590
3540 IF COUPS<8 THEN PRINT "-PASSABLE";
3550 IF COUPS>7 THEN PRINT "-TRES MAUVAIS";
3590 PRINT
3595 RETURN
```

```
4000 REM REGLES DU JEU
4010 SLOW
4020 PRINT TAB 8;"PARTIE DE GOLF"
4030 PRINT ,,"VOULEZ VOUS LES REGLES DU JEU?"
4040 RAND
4050 INPUT Y$
4060 IF CHR$ CODE Y$<>"Y" THEN RETURN
4100 PRINT ,,"LA BALLE EST REPRESENTEE PAR UN",
         "X, LE TROU PAR UN 0. VOUS DEVEZ",
         "ESTIMER LA DIRECTION ET LA",
         "FORCE DU JET DE LA BALLE."
4110 PRINT "  COMME LES AIGUILLES D UNE",
         "MONTRE, LES DIRECTIONS VONT DE",
         "1 A 12. L EMPLOI DE VALEURS",
         "DECIMALES (EX:2.6) EST POSSIBLE."
4120 PRINT "  IL FAUT UNE FORCE DE 60 POUR",
         "LANCER LA BALLE D UN COTE A",
         "L AUTRE, DE 36 POUR LANCER LA",
         "BALLE DE HAUT EN BAS."
4130 PRINT "  APPUYER SUR UNE TOUCHE POUR",
         "JOUER UNE NOUVELLE FOIS."
4140 PRINT ,,"AMUSEZ VOUS BIEN."
4150 PRINT ,,"(PRESSEZ UNE TOUCHE)"
4160 PAUSE 4E4
4170 POKE 16437,255
4180 RETURN
9900 SAVE "GOLF"
9910 RUN
```

Voici les modifications à apporter pour exécuter le programme sur un ZX80 :

```
605 PAUSE 150

840 PAUSE 4E4
841 POKE 16437,255
```

# JEUX DE CARACTÈRES

***BUT***

Ce sous-programme permet d'afficher un texte en caractères géants (format 8 x 8) sur l'écran. Il fait appel à l'instruction PLOT pour dessiner les lettres ; on peut ainsi "caser" huit caractères par ligne.

L'écran ne comportant que 44 éléments d'image, vous avez droit en tout à cinq lignes (44/8=5 reste 4) ; les quatre éléments restants peuvent servir à ménager un blanc entre les lignes géantes.

Imaginez l'effet produit en présentant de cette façon un programme dont on lance l'exécution : le titre du programme apparaît alors en lettres géantes sur votre écran !

***FONCTIONNEMENT***

Ce sous-programme fait appel à la mémoire morte ROM 8K contenue dans le ZX81 (un générateur de caractères est situé à partir de l'adresse 7680). Bien qu'intéressantes, les explications contenues dans les paragraphes suivants ne sont pas strictement indispensables pour la compréhension du programme lui-même. Le lecteur ne connaissant pas encore toutes les ficelles d'un microprocesseur Z80 peut donc choisir d'en faire abstraction pour le moment.

Le générateur de caractères est une partie de la mémoire contenant le "dessin" de chaque caractère destiné à l'affichage sur écran. Chaque caractère affiché occupe une place mémoire de huit octets ; chaque octet correspond au dessin d'une rangée de points sur l'écran. Prenons par exemple la lettre "A". L'adresse du dessin générateur de la lettre A est 7984 (voir ci-dessous pour plus de détails). Les huits octets placés à partir de cette adresse contiennent les valeurs suivantes :

| adresse | contenu décimal | contenu binaire | affichage sur l'écran |
|---|---|---|---|
| 7984 | 0 | 00000000 | ........ |
| 7985 | 60 | 00111100 | ..****.. |
| 7986 | 66 | 01000010 | .*....*. |
| 7987 | 66 | 01000010 | .*....*. |
| 7988 | 126 | 01111110 | .******. |
| 7989 | 66 | 01000010 | .*....*. |
| 7990 | 66 | 01000010 | .*....*. |
| 7991 | 0 | 00000000 | ........ |

Vous pouvez reconnaître la forme de la lettre A dans la dernière colonne.

En règle générale, chaque dessin de caractère est obtenu à l'adresse calculée de la façon suivante :

7680 + (code caractère * 8)

... attention ! Ceci n'est valable que pour les caractères possédant un code compris entre 0 et 63. Tous les autres codes (codes supérieurs à 63) n'ont pas de signification ou correspondent à des combinaisons de caractères. Prenons par exemple le mot-clé RND : son code (64) donnera l'affichage simultané des lettres R, N et D.

## LE SOUS-PROGRAMME

Que peut-on dire de ce sous-programme ?
Il utilise le générateur de caractères tel que nous l'avons décrit ci-dessus pour dessiner des points sur l'écran. Le générateur occupant une matrice de 8 x 8, vous comprenez aisément comme celle-ci peut être agrandie.
Il y a cependant un problème à surmonter : comment connaître le contenu des bits à l'intérieur du générateur ? Le sous-programme dessine un point si le bit est à 1 et n'en dessine pas si le bit est à 0 (dans le tableau ci-dessus, nous avons représenté le bit à l'état 1 par un astérisque et le bit à l'état 0 par un point).

Nous avons essayé deux versions de ce sous-programme dont aucune n'est particulièrement rapide ; cependant, dans le deuxième cas le temps d'exécution est sensiblement plus court. En revanche, la première version a le mérite d'être plus facile à suivre, puisqu'elle utilise la méthode conventionnelle pour isoler un bit d'un octet.

Avant d'appeler le sous-programme, il convient de définir trois variables (nous rencontrerons plus loin un exemple d'application de la routine à l'intérieur d'un programme) :

**A$** variable alphanumérique contenant la chaîne de caractères à afficher en grand format. Elle comprend huit caractères au maximum (sinon le ZX81 arrête l'exécution en affichant le code erreur B). Les caractères doivent obligatoirement avoir un code compris entre 0 et 63.

**XX** représente les coordonnées horizontales de l'élément d'image correspondant au premier caractère de la chaîne.

**YY** représente les coordonnées verticales de l'élement d'image correspondant au premier caractère de la chaîne. Remarquez que le chiffre 43 représente le haut de l'écran alors que le chiffre 0 représente le bas.

***Version 1 - la plus facile (?) à comprendre mais la plus lente***

```
9300 REM DESSINE A$ EN CARACTERES GEANTS
9305 FOR A=1 TO LEN A$                   (pour chaque caractère
9310 LET XC=CODE A$(A)                   (donne le code du caractère
9315 GOSUB 9340                          (imprime en 8 x 8
9320 LET XX=XX+8                         (position suivante sur l'écran
9325 NEXT A                              (lettre suivante
9330 RETURN                              (sortie-exécution terminée
9340 FOR Y=YY TO YY-7 STEP -1            (8 rangs de haut en bas
9345 LET RANG=PEEK (7680+8*XC+YY-Y)      (cherche dessin du rang dans ROM
9350 LET SHR=128                         (valeur initiale du masquage de bit
9355 FOR X=XX TO XX+7                    (pour chaque bit du rang
9360 LET Z=INT (RANG/SHR)                (met état du bit dans Z
9365 LET RANG=RANG-Z*SHR                 (écarte ce bit
9370 LET SHR=SHR/2                       (donne masquage du bit suivant
9375 IF Z THEN PLOT X,Y                  (dessine si bit est à 1
9380 NEXT X                              (dessin du bit suivant
9385 NEXT Y                              (rang suivant
9390 RETURN                              (caractère complet
```

La version 2 est basée sur le même principe. Les calculs effectués par le ZX81 étant lents, cette deuxième méthode emploie au maximum les chaînes de caractères (utiliser la virgule flottante sur cinq octets au lieu de se contenter des valeurs entières revient exactement au même que d'assommer une mouche avec un pilon !). Ici, le masquage des bits n'est plus obtenu par l'intermédiaire d'une variable numérique, mais grâce à une chaîne de caractères dont les codes représentent les bits à masquer : 128, 64, 32, 16, 8, 4, 2, 1.

Cela implique que la mise en pratique est plus facile avec des caractères qu'avec des nombres ; en réalité, l'exécution n'en est pas accélérée pour autant.

***Version 2 - plus rapide mais pleine d'embûches***

```
9300 REM DESSINE A$ EN CARACTERES GEANTS
9305 LET Z$="           "                (voir plus bas
9310 FOR A=1 TO LEN A$
9315 LET XC=CODE A$(A)
9320 GOSUB 9340
9325 LET XX=XX+8
9330 NEXT A
9335 RETURN
9340 FOR Y=YY TO YY-7 STEP -1
9345 LET X$=CHR$ (PEEK (7680+8*XC+YY-Y))
9350 FOR X=1 TO 8
9355 IF X$<Z$(X) THEN GOTO 9370
9360 PLOT XX+X-1,Y
9365 LET X$=CHR$ (CODE X$-CODE Z$(X))
9370 NEXT X
9375 NEXT Y
9380 RETURN
```

Dans le sous-programme présenté ci-dessus, la ligne 9305 met dans la variable Z$ la chaîne de caractères permettant le masquage des bits. Les caractères de cette chaîne sont :

| | | |
|---|---|---|
| 128 | espace "inversé" | GRAPHICS/SPACE |
| 64 | **RND** | FUNCTION/T |
| 32 | 4 | 4 |
| 16 | ( | ( |
| 8 | carré gris | GRAPHICS/SHIFT/A |
| 4 | 1/4 de carré | GRAPHICS/SHIFT/4 |
| 2 | " " | GRAPHICS/SHIFT/2 |
| 1 | " " | GRAPHICS/SHIFT/1 |

Cette chaîne de caractères qui semble plutôt bizarre (elle n'est pas utilisée de façon habituelle) nous permet d'éviter l'utilisation des opérateurs de BOOLE et des variables entières.

***EXEMPLE D'APPLICATION***

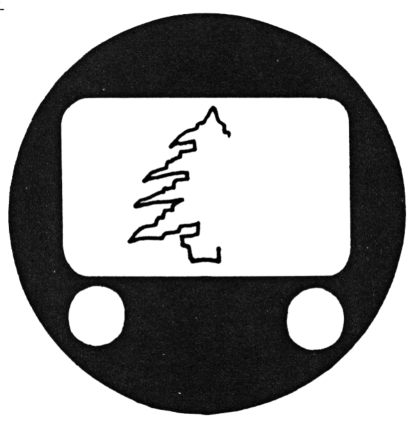
Dans quel contexte utiliser ce sous-programme ?
Voici un exemple d'utilisation. Le message HELLO va être imprimé en lettres géantes au centre de l'écran :

```
100 REM IMPRIME HELLO EN CARACTERES GEANTS
110 LET A$="HELLO"        (message à imprimer
120 LET XX=12             (coordonnée horizontale
130 LET YY=26             (coordonnée verticale
140 GOSUB 9300            (appelle le sous-programme
150 STOP                  (voyez le résultat !
```

Il va de soi qu'une des deux routines présentées plus haut (de préférence la seconde) doit être ajoutée au programme pour que celui-ci puisse fonctionner.

## L'ARDOISE MAGIQUE

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***



Peut-être était-ce l'un de vos jouets favoris quand vous étiez petit ? Nous vous proposons ici une variante qui vous permettra de réussir quelques esquisses intéressantes.

Un seul inconvénient : l'ordinateur (pas plus que le jouet) n'est capable de dessiner des courbes parfaites.

Utilisez les touches de déplacement du curseur "5" et "8" (sans appuyer sur la touche SHIFT), pour dessiner un trait dans n'importe quelle direction. Pendant que vous appuyez sur la touche, le trait se dessine.

A tout moment, vous pouvez appuyer sur la touche "Ø" (libellée RUB OUT, de l'anglais gommer) pour effacer l'écran ; le curseur reste alors à la même place. Si vous désirez par exemple démarrer votre esquisse du coin en haut et à droite, utilisez le processus suivant : déplacez le curseur jusqu'à l'endroit choisi comme point de départ puis appuyez sur la touche "Ø" pour effacer l'écran.

Notez le contenu de la ligne 30 du programme : celle-ci consulte la variable-système RAMTOP pour voir si l'extension 16K RAM est en place. Sans cette extension, la variable contient la valeur 68 (68*256=17408, nombre correspondant à l'adresse du dernier pas de mémoire - 16384+1024). Si une valeur supérieure à 68 est détectée il convient de brancher l'extension mémoire 16K ou 4K (seule l'extension 4K RAM était vendue avec le ZX80 au début de sa commercialisation).

Muni de l'extension mémoire, vous pouvez occuper sans problème la totalité de l'écran. Dans le cas contraire, il convient de délimiter une portion plus restreinte de l'écran pour éviter l'affichage du code erreur 4.

```
 10 LET MX=64                          (largeur maximum de l'écran
 20 LET MY=44                          (hauteur maximum de l'écran
 30 IF PEEK 16389<69 THEN LET MY=23(extension 16K ?
 40 LET X=NOT MX                       (point de départ = 0.
 50 LET Y=MY-1                         (point de départ = MY
100 LET D=CODE INKEY$-28               (donne le code de la touche
110 LET DX=(D=5)*-1+(D=8)              (calcule direction X
120 LET DY=(D=6)*-1+(D=7)              (calcule direction Y
130 IF NOT D THEN CLS                  (efface si touche pas appuyée
140 IF INT ((X+DX)/MX) OR INT ((Y+DY)/MY) THEN GOTO 200
                                       (sur l'écran ?
150 LET X=X+DX                         (modifie position X
160 LET Y=Y+DY                         (modifie position Y
170 PLOT X,Y                           (dessine un nouvel élément
200 GOTO 100                           (continue...
```

*Ceux qui possèdent le ZX80 équipé de la mémoire morte ROM 8K* doivent ajouter les trois lignes suivantes pour pouvoir observer l'écran pendant l'exécution du programme :

```
135 IF NOT DX AND NOT DY THEN GOTO 210   (pas de touche appuyée

210 PAUSE 4E4                          (affichage écran
220 GOTO 100
```

Modifier également la ligne 30 de la façon suivante :

```
30 IF PEEK 16389<69 THEN LET MY=22
```

Tant que vous appuyez sur une touche, l'écran reste blanc ; dès que vous relâchez cette touche, votre oeuvre d'art apparaît sur l'écran !

## CREATION D'UN FICHIER D'INDIVIDUS

***CAPACITE MEMOIRE NECESSAIRE : 16K RAM***

Si vous aimez les jeux de guerre et que vous ayez des problèmes de figurants, le programme suivant vous conviendra parfaitement.



Chercher des noms différents pour chaque personnage est un travail peu intéressant où l'innovation figure rarement. Vous allez être sauvé par ce programme qui crée tout un fichier de noms d'individus (plus ou moins recommandables !) avec, en prime, leur activité et leurs particularités (Quotient Intellectuel (ou QI), FIDélité, FORce, DEXtérité, COMbativité, FIAbilité et RIChesse). Que désirer de plus ?

La structure de ce programme permet une adaptation rapide à tous les jeux. Les lecteurs préférant jouer aux cow-boys et aux indiens modifieront aisément les chaînes alphanumériques contenant les différents critères.

Ceux qui possèdent une imprimante sont nettement avantagés ; ils peuvent imprimer les caractéristiques de tel ou tel personnage intéressant en utilisant la commande COPY. Vous pouvez à votre gré garder ou éliminer les fiches des personnages qui ne vous conviennent pas.

Les paramètres grandeur et poids de chaque personnage sont calculés d'après des tables figurant dans le sous-programme d'initialisation. Le programme contrôle la grandeur et le poids pour éviter de créer un nain de cent kilos ou un géant de cent grammes !

La sélection du type d'activité des personnages créés est liée à leurs particularités (toujours pour ne pas créer de non-sens). La probabilité de créer un Sorcier-Guerrier est assez faible.

Tous les noms des personnages contenus dans la variable alphanumérique F$ sont séparés par une barre oblique (/). Les surnoms de ces personnages sont stockés de la même manière dans S$. Deux autres variables NF et NS contiennent respectivement le nombre de noms et de surnoms. Si vous désirez ajouter d'autres noms à la liste

proposée, n'oubliez pas de modifier le contenu des variables NF et NS en conséquence (lignes 9310 et 9320).

```
   1 REM FICHIER PERSONNAGES
   2 REM ******************
  10 FAST
  20 DIM T$(4,14)                            (types de personnages
  30 LET T$(1)="GUERRIER"
  40 LET T$(2)="SORCIER"
  50 LET T$(3)="VOYOU"
  60 LET T$(4)="SORCIER-GUERRIER"
  70 LET A$="FORQI.FIDDEXCOMFIARIC"         (caractéristiques
  80 RAND
  90 GOSUB 9300                              (initialise le tableau
 100 LET COTE=0                              (réinitialise la cote des
                                                      personnages
 110 DIM A(9)                                (caractéristiques
 120 FOR X=1 TO 8                            (crée 8 caractéristiques
 130 GOSUB 8100                              (jette 3 dés
 140 IF X>3 THEN GOTO 170                    (contrôle les 1ères carac-
                                                      téristiques
 150 IF A(X)>12 THEN LET COTE=COTE+A(X)-12 (incrémente la cote
 160 IF A(X)<9 THEN LET COTE=COTE+A(X)-9   (ou la décrémente
 170 NEXT X
 180 LET X=9                                 (génère le poids
 190 GOSUB 8100                              (jette 3 dés
 200 IF ABS (A(8)-A(9))>3 THEN GOTO 180     (assure compatibilité hau-
                                                      teur/poids
 210 LET T=1                                 (c'est un guerrier
 220 FOR X=1 TO 5                            (est-ce un sorcier-guerrier ?
 230 IF A(X)<12 THEN GOTO 270
 240 NEXT X
 250 LET T=4                                 (ciel ! un sorcier-guerrier
 260 GOTO 500
 270 IF A(2)<10 OR A(5)<8 OR A(1)<8 THEN GOTO 290
                                             (sorcier nécessite QI.>=10
 280 IF A(2)>A(3) AND A(2)>A(1) THEN LET T=2 (sorcier
 290 IF A(1)>A(2) AND A(1)>A(3) THEN LET T=1 (guerrier
 300 IF A(3)>A(1) OR A(3)>A(2) THEN LET T=3  (voyou
 500 LET A(7)=A(7)*10                        (richesse
 510 CLS
 520 LET A(8)=A(8)-2                         (hauteur = 1 à 16
 530 LET A(9)=A(9)-2                         (poids = 1 à 16
```

```
600 GOSUB 8000                                  (crée et imprime le nom
610 PRINT
620 PRINT "TYPE:";T$(T)
630 FOR X=0 TO 6
640 PRINT A$(X*3+1 TO X*3+3);"  :";A(X+1)          (caractéristiques
650 NEXT X
660 PRINT "HAUT:";INT (H(A(8))/12);"PIEDS";
      H(A(8))-INT (H(A(8))/12)*12;"POUCE(S)",     (hauteur
670 PRINT "PDS :";W(A(9));"LIVRES"             (poids
680 PRINT
690 PRINT "COTE:";COTE                         (cote du personnage
700 PAUSE 250                                  (présente le personnage
710 POKE 16437,255
720 IF INKEY$="" THEN GOTO 100                 (autre personnage si pas de tou-
                                                            che appuyée
730 STOP                                       (arrêt momentané
740 GOTO 100                                   (crée un nouveau personnage
8000 REM CHOIX DU NOM
8010 LET N=INT (RND*NF)+1
8020 LET D$=F$
8030 GOSUB 9000                                (extrait le nom N de D$
8040 PRINT "NOM :";W$;" ";                     (affiche ce nom
8050 LET N=INT (RND*NS)+1                      (choisi un surnom
8060 LET D$=S$
8070 GOSUB 9000
8080 PRINT W$                                  (affiche le surnom
8090 RETURN
8100 REM JETTE 3 DES
8110 LET A(X)=0                                (valeur initiale
8120 FOR Y=1 TO 3
8130 LET A(X)=A(X)+INT (RND*6)+1               (additionne les valeurs
8140 NEXT Y
8150 RETURN
9000 REM MET N-IEME MOT DE D$ DANS W$          (voir chapitre 3
9005 LET XW=0
9010 LET W$=""
9015 FOR X=1 TO N-1
9020 LET XW=XW+1
9025 IF XW>LEN D$ THEN RETURN
9030 IF D$(XW)<>"/" THEN GOTO 9020
```

```
9035 NEXT X
9040 LET XW=XW+1
9045 IF XW>LEN D$ THEN RETURN
9050 IF D$(XW)="/" THEN RETURN
9055 LET W$=W$+D$(XW)
9060 GOTO 9040
9300 REM LISTE DES NOMS                (Sous-programme d'initialisation
9301 REM NF=NOMBRE DE NOMS
9302 REM NS=NOMBRE DE SURNOMS
9304 REM F$=NOMS
9305 REM S$=SURNOMS
9310 LET NF=15
9320 LET NS=25
9330 LET F$="ARTHUR/PIERRE/PAUL/YVAN/JEAN/
           TOMMY/MAX/PIERROT/JULES/TREVOR/
           BOB/JO/ERIC/DEDE/JEHAN"
9340 LET S$="LE ROUGE/LE TOUT-PUISSANT/LE PIEUX/LE CRUEL/
           LE RENARD/LE BAGARREUR/LE SIMPLET/LE VOLEUR/
           LA TERREUR/LE GRAND/LE MAGNIFIQUE/GAGNEPETIT/
           LE FOU/SANS PEUR/LE BRAVE/LE SANGUINAIRE/
           DE NANTERRE/LE VAGABOND/L ENRAGE/LE TIMIDE/
           L EFFRONTE/OEIL DE TAUPE/LE TRICHEUR/
           NEZ CASSE/LE TERRIBLE"

9350 DIM H(16)                         (Crée la table de hauteur
9360 FOR X=1 TO 16
9370 LET H(X)=INT (48.5+((X-1)*2.5))
9380 NEXT X
9390 DIM W(16)                         (Crée la table de poids
9400 LET W$="075090105120135150160170180190200225250280310350"
9410 FOR X=1 TO 16
9420 LET W(X)=VAL (W$((X-1)*3+1 TO (X-1)*3+3))
9430 NEXT X
9500 CLS
9510 PRINT "FICHIER PERSONNAGES"
9515 PRINT
9520 PRINT "CHAQUE PERSONNAGE VOUS EST ",
           "PRESENTE QUELQUES SECONDES. SI",
           "VOUS DESIREZ ETUDIER UN INDIVIDU",
           "PLUS LONGTEMPS, PRESSEZ UNE ",
           "TOUCHE CELA ARRETE LE",
           "PROGRAMME (SI VOUS AVEZ UNE",
           "IMPRIMANTE, UTILISEZ LA",
           "COMMANDE COPY)."
9540 PRINT ,,"(PRESSEZ UNE TOUCHE)"
9550 PAUSE 40000
9560 POKE 16437,255
9590 RETURN
```

# TRUCS ET ASTUCES

Ce chapitre a pour but de présenter quelques "trucs" glanés ça et là au hasard des programmes. En réalité, il n'est question ni d'auxiliaires indispensables de la programmation, ni d'astuces époustouflantes pour sauvegarder de la place en mémoire.

Ces quelques idées toutes simples ont juste le souci de rendre la programmation parfois plus agréable, particulièrement sur le ZX81.

## *PAUSE INFINIE*

Vous rencontrerez souvent l'utilisation de cette astuce dans les exemples de programme proposés.

D'après la notice Sinclair, l'emploi de l'instruction PAUSE associée à une valeur numérique supérieure à 32767 déclenche une pause qui se poursuit à l'infini. L'exécution de la suite du programme est alors obtenue en appuyant sur une touche quelconque.

Cette instruction est particulièrement utile pour les utilisateurs de ZX80 équipés de la ROM 8K ; en effet, ce dispositif ne permet pas l'affichage sur écran pendant l'exécution d'un programme. Aussi, toutes les possibilités de l'instruction INKEY$ ne peuvent-elles être utilisées. Cette remarque ne concerne pas le ZX81, sauf si le programme est exécuté en mode rapide.

Dans ce cas, le programme devra contenir une ligne du type :

```
200 PAUSE 33000
```

Comme pense-bête (pas si bête que ça !), on peut utiliser l'expression :

```
200 PAUSE 4E4
```

... ceci pour deux raisons : cette écriture occupe un peu moins de place en mémoire et constitue un moyen mnémotechnique (pour les anglophiles ! Four-i-four≈Forever).

Souvenez-vous que la notation scientifique est tout à fait autorisée avec le Basic du ZX81 ; l'expression 4E4 correspond à 40000 (chiffre supérieur à 32767).

## *FIN D'UN PROGRAMME*

Il est question ici d'un autre de nos "dadas" : en effet, il est souvent préférable de terminer un programme avec un message de reconnaissance, plutôt qu'avec un 0/375 affiché en bas de l'écran.

Pour cette raison, la ligne :

```
9999 STOP
```

est insérée à la fin d'un programme ; quel que soit le moment choisi pour terminer le programme, il suffit de taper :

```
... GOTO 9999
```

Quand le programme rencontre cette ligne, le message 9/9999 s'affiche en bas de l'écran ; celui-ci indique que tout est "O.K.". Tout autre code erreur ressort plus facilement quand vous attendez l'affichage de 9/9999.

La plupart des programmes menés en mode conversationnel (INPUT...) sont interrompus purement et simplement en appuyant sur la touche BREAK. Ceci a pour conséquence l'affichage du code erreur D, comme vous pouvez vous y attendre.

## *INSTRUCTION REM*

Un des plus gros problèmes à surmonter quand on reprend les programmes de quelqu'un d'autre est de comprendre leur structure. On peut passer un temps fou à essayer de "saisir" un passage complexe.

Une grande partie des programmes que vous avez écrits, ou que vous écrirez, est destinée à l'oubli. Dans le meilleur des cas, ceux-ci seront remplacés par des versions améliorées.

Pourquoi ? Parce que la plupart sont conçus pour tester une idée et ne méritent pas une perte de temps importante. C'est une autre histoire si vous écrivez des programmes destinés à durer ; peut-être même en donnerez ou en vendrez-vous à des amis.

Il devient alors très important de passer du temps à préciser les fonctions de chaque étape du programme pour que celui-ci soit plus facile à comprendre.

L'instruction REM permet d'annoter les différentes étapes d'un programme, ce qui le rend plus facile à suivre. L'emploi de cette instruction est conseillé dans les situations suivantes :

a- au début de chaque programme pour définir la finalité de celui-ci,
b- à chaque portion "logique" du programme (boucle principale, initialisation, section finale, etc...),
c- avant chaque passage particulièrement complexe du programme afin de souligner brièvement son fonctionnement.

Attention ! Avec le ZX81 1K RAM, la place en mémoire est comptée ; elle ne permet pas cette consommation superflue d'octets. L'écriture de la plupart des programmes nécessite un déploiement important d'astuces pour tout "caser" dans moins d'1K de mémoire, alors pour ce qui est des annotations...

Nous avons essayé de résoudre ce problème en émaillant les programmes présentés de commentaires adéquats.

Si votre programme vous le permet, essayez quand même d'utiliser une instruction REM.

Si l'extension 16K RAM est branchée, les économies de place en mémoire rentrent moins en considération. Vous pouvez vous permettre d'être prolixe avec les instructions REM ; cela ne prend que quelques minutes supplémentaires pour les entrer et n'a pas d'effet sur le déroulement du programme.

Remarquez que dans la plupart des programmes présentés dans ce livre, les instructions REM sont réduites à leur strict minimum ; ceci non seulement pour ne pas faire double emploi avec les annotations parallèles, mais encore pour éviter un volume de frappe trop important.

## CASSETTES

Au cas où vous ne le sauriez pas encore, des cassettes C 12 sont disponibles dans la plupart des boutiques informatiques. Ce type de cassettes est beaucoup plus adapté aux besoins de la programmation que les C 60 ou C 90.

En effet, il peut être extrêmement ennuyeux d'attendre que la cassette ait fini de s'enrouler d'un bout à l'autre de la face ; plus la durée de la cassette est courte, moins vous avez à patienter.

Peut-être serez-vous tenté de stocker tous vos programmes sur une même cassette ? Résistez à la tentation ; trois ou quatre programmes par face sont largement suffisants. Le stockage d'une quantité supérieure peut donner lieu à une perte de temps importante si le programme désiré est le dernier.

Il est beaucoup plus agréable d'avoir une "programmathèque" rangée et étiquetée soigneusement. Il faut ménager ses effets auprès d'éventuels spectateurs ! Rien n'est plus énervant, lors

de la présentation d'un programme à un ami, que de s'excuser avec des *"il y en a pour une minute"* ou *"j'y suis presque"* !

Le modèle original du ZX80 ne permettait pas la sauvegarde ou le chargement d'un programme accompagné d'un libellé. De ce fait, combiner plusieurs programmes sur une même face de cassette relevait de l'acrobatie. A juste titre, on conseillait alors aux utilisateurs de ZX80 de sauvegarder un seul programme par face.

Ceux qui ont acquis l'extension ROM 8K ont maintenant la possibilité de "nommer" leurs programmes. Ils peuvent ainsi les grouper sur une seule face en conservant une certaine facilité d'accès.

## SAUVEGARDE DES VARIABLES

Pour sauvegarder un programme comportant plusieurs variables pré-affectées, il faut insérer un sous-programme "chargement-exécution" dans le programme principal. Celui-ci sera automatiquement exécuté dès que le chargement en mémoire (à partir de la cassette) sera terminé. Ce procédé évite de taper RUN par erreur et d'effacer toutes les variables consciencieusement "mises à l'abri".

Un exemple de sous-programme "chargement-exécution" se trouve à la page 110 du manuel Sinclair.

Vous rencontrerez un autre exemple dans ce livre avec le "sous-programme de chargement" du "Jeu de l'aventure" (page 115 ).

## CHARGEMENT DES PROGRAMMES

Dans les débuts du ZX80, une grande quantité de propos contradictoires furent échangés à propos des problèmes inhérents au chargement des programmes. Certains trouvaient cette opération très hasardeuse, alors que d'autres ne se heurtaient à aucun problème particulier.

Comme dans bien des cas, si vous respectez soigneusement les instructions du chapitre 16 de la notice Sinclair, vous serez assuré une fois la mise au point effectuée de la fiabilité de cette opération.

Vous avez peut-être découvert à vos dépens un problème qui n'a pas été signalé par le fabricant : l'enregistrement des programmes sur un magnétophone stéréo amène souvent des anomalies.

Si les têtes du magnétophone stéréo servant à l'enregistrement sont mal alignées, un décalage de phase peut se créer entre les deux canaux quand on utilise un matériel différent pour la lecture (ou le même matériel si l'alignement est particulièrement mauvais).

Vos oreilles ne décèleront rien ; en effet, celles-ci sont incapables de détecter le plus petit décalage de phase. Par contre, votre ZX81 en est parfaitement capable !

Le remède est simple : si vous utilisez un système stéréo, enregistrez sur un seul canal (de préférence le canal de gauche qui est le canal extérieur, ce qui réduit les interférences). Vous réaliserez probablement vous-même qu'il est plus simple d'utiliser un enregistreur portable bon marché.

## *A PROPOS DES CHAINES DE CARACTERES*

Le Basic du ZX81, aussi complet soit-il, souffre de quelques petites imperfections. Une de celles-ci est l'affichage systématique (ô combien embêtant !) du code erreur 3 si le programme essaye d'isoler une partie inexistante d'une chaîne de caractères.
Les deux lignes :

```
LET A$="ABCDE"
PRINT A$(6)
```

provoqueront l'affichage du code erreur 3. En effet, la variable alphanumérique A$ ne contient que cinq caractères. Il aurait été préférable d'avoir l'affichage d'une "chaîne vide", mais on ne peut pas tout avoir.

Si votre programme est écrit pour extraire le premier caractère d'une chaîne, l'erreur 3 peut être évitée en tapant :

```
CHR$ CODE A$
```

En effet, l'instruction **CODE** fournit le code du premier caractère. Si la chaîne est "vide", le code obtenu est zéro.

Dans ce cas la ligne suivante devient inutile :

```
IF LEN X$=0 THEN ....
```

Si vous désirez obtenir un caractère particulier à l'intérieur d'une chaîne, introduisez la ligne :

```
CHR$ CODE X$(n TO )
```

Celle-ci vous évitera l'affichage du code erreur 3 si n>=1.

## *ECONOMISONS LA MEMOIRE*

Le chapitre "Optimisation des programmes" vous a déjà sensibilisé sur ce problème primordial ; celui-ci mérite quelques approfondissements supplémentaires.
Une ligne du type :

```
IF xxxx <> 0 THEN ....
```

peut être avantageusement remplacée par :

IF xxxx THEN ....

En effet, le ZX81 considère une valeur différente de zéro comme étant "vraie".

De même une ligne du type :

IF xxxx = 0 THEN ....

peut être remplacée par :

IF **NOT** xxxx THEN ....

Ces deux méthodes ralentissent l'exécution d'un programme mais permettent d'économiser de la place en mémoire.

Ne perdez pas de vue qu'il s'agit de tirer le maximum des possibilités du ZX81 et non de gagner un concours récompensant le programme le plus compliqué !
Un autre ordinateur ne comprendrait pas obligatoirement les lignes présentées ci-dessus. Aussi, soyez prudent avant d'inférer que votre programme peut fonctionner sur n'importe quel ordinateur.

## ***ENTREE DES EXPRESSIONS***

Le ZX81 possède une particularité importante qu'il ne faut pas négliger : n'importe quelle expression valable (c'est-à-dire du type demandé) peut être entrée par l'intermédiaire de l'instruction INPUT.
Cette particularité est mise à profit dans le programme "calcul d'un Ecart-Type" où le mot FIN est entré derrière les données numériques pour signifier la fin de la liste.
Comment est-il possible d'entrer un mot alors que l'ordinateur attend des valeurs numériques ?

La valeur 1E38 ayant été préalablement affectée à la variable FIN au cours du programme, celle-ci est acceptée par l'instruction INPUT en tant que valeur numérique (la valeur 1E38 a peu de chance d'être entrée en tant que donnée).
Le ZX81 serait tout aussi satisfait avec une valeur telle que 25*3.555/6.

Le même procédé est applicable aux chaînes de caractères. Toute variable alphanumérique préalablement affectée peut être entrée par l'intermédiaire de l'instruction INPUT. Dans ce cas, il est nécessaire de "gommer" les guillemets que le ZX81 met d'office pour les opérations d'entrée de chaîne. Le meilleur moyen consiste à appuyer sur la touche EDIT (SHIFT/1) qui efface toute donnée

venant d'être tapée.

## *UTILISATION DE L'EXTENSION 16K*

La plupart des programmes présentés dans ce livre ont été conçus et expérimentés avec l'extension 16K RAM, puis éventuellement adaptés à la version 1K RAM.

Malheureusement, un programme créé avec l'extension 16K et sauvegardé en conséquence ne peut être chargé sur la version 1K sans difficulté. En effet, la routine de gestion de l'écran fixe le nombre de lignes affichées à 24 quand le ZX81 possède plus de 3.75K de mémoire utilisateur ; pour cette raison, un programme occupant juste 1K sur la version 16K RAM ne peut être chargé sur la version de base du ZX81.

Si vous désirez plus tard utiliser la version de base pour faire tourner un programme mis au point sur 16K, prenez la précaution avant de le sauvegarder sur cassette de taper en mode direct les commandes suivantes :

```
POKE 16388,0
POKE 16389,68
CLS
```

Les deux premières lignes fixent la variable système RAMTOP à la valeur 17408 (pour la version 1K). La commande CLS sert à réduire au maximum le nombre de lignes gérées sur l'écran.

Ces quelques précautions étant prises, votre programme devrait être maintenant utilisable sur le ZX81 version 1K ; sinon, c'est un autre problème.

## *PRECISION DES CALCULS*

Comme la plupart des ordinateurs, le ZX81 est incapable de manipuler des nombres sans les entacher d'erreur.

L'homme compte en base 10 pour l'excellente raison que les doigts de la main constituent la meilleure machine à calculer qu'il ait jamais inventée.

Cependant, le système décimal ne permet pas la représentation exacte de certains nombres, par exemple $\pi$ (PI) ou racine de 2 (SQR(2)). Ces nombres sont dits <u>irrationnels</u>.

L'ordinateur a le même problème, bien que les nombres dont il ne peut donner la représentation exacte soient de nature différente ;

en effet, dans son cas, les calculs sont effectués en base 2 et non en base 10.

Ceci provoque l'apparition intéressante (mais souvent gênante) d'erreurs dans la partie fractionnaire du résultat des opérations (principalement dans le cas de la division). La commande PRINT arrondit à l'occasion ces petites erreurs ; méfiez-vous quand même de certaines énormités qui peuvent se glisser dans vos calculs au moment où vous vous y attendez le moins.

Pour vous en convaincre, faites exécuter les deux lignes suivantes :

```
LET V=100*0.15
PRINT V,INT,V
```

Le contenu logique que l'on est en droit d'attendre pour la variable V est égal à 15, ce qui se vérifie. On pourrait s'attendre à obtenir la même valeur après séparation de la partie entière. Surprise ! C'est le nombre 14 qui est affiché.

En effet, l'instruction PRINT arrondit la valeur réelle de V qui est 14.9999...

Vous pensez probablement que l'imprécision du calcul va être mise en évidence par l'exécution de la ligne suivante :

```
PRINT V-INT V
```

peine perdue, car l'instruction PRINT arrondit de nouveau le résultat (0.99999999) et affiche la valeur 1.

Dans ce cas particulier l'erreur peut être évitée en écrivant l'opération sous la forme :

```
LET V=100*15/100
```

Le remplacement du nombre décimal (0.15) par la fraction équivalente (15/100) donne une précision accrue puisque les nombres mis en jeu sont plus importants.

Si vous êtes amené à écrire des programmes de calculs financier, statistique ou mathématique, ces petits ennuis deviennent vite de vrais casse-tête. Chacune de ces erreurs de précision ayant son remède particulier, il est difficile de donner une solution universelle.

Un conseil cependant : les résultats obtenus sont généralement plus précis si vous *veillez dans la mesure du possible à ce que les multiplications précédent les divisions*. Cette méthode est parfois employée avec les calculatrices pour éviter une perte de précision ; en cela, le ZX81 n'est pas bien différent.

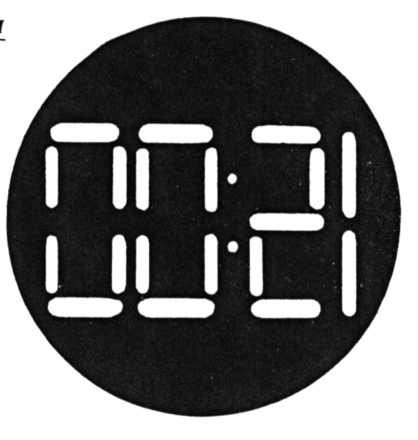
## *** RECREATION ***

Voici un tout petit programme qui se contente de dessiner de jolis motifs. La version ZX81 1K RAM est amplement suffisante.

```
10 FOR X=0 TO 9999
20 SCROLL
30 LET Z=SIN (X*PI/10)*15
40 PRINT AT 21,16-Z;"*";AT 21,16+Z;"*"
50 NEXT X
```

## L'HORLOGE NUMERIQUE

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***



Ce programme évite les embûches du programme HORLOGE présenté dans le manuel Sinclair. La plus importante concerne le réglage très délicat de cette horloge. En effet la modification de la ligne d'instruction PAUSE 42 ne provoque que des sauts de 1/50ème de seconde à la fois. Bien que des lignes neutres (c'est-à-dire sans influence sur le déroulement) puissent être insérées dans le programme pour créer un petit laps de temps supplémentaire, on n'obtient jamais un multiple exact de 1/50ème de seconde et l'horloge n'est jamais à l'heure.

Peut-on espérer obtenir une précision mirobolante de ce type d'horloge ? Pas tout à fait. La version présentée ci-dessous simule une horloge beaucoup plus précise que celle obtenue par le programme Sinclair mais présente l'inconvénient de travailler en mode lent.

Les propriétaires d'un ZX80 (modifié avec la ROM 8K) ne peuvent donc pas faire fonctionner ce programme (désolé). Ne désespérez pas, la méthode employée dans ce programme peut être mise à profit dans d'autres cas de figures qui ne nécessitent pas l'utilisation continue de l'écran.

En quoi consiste cette méthode ? Elle est basée sur la variable système FRAMES qui est décrémentée à chaque "rafraîchissement" de l'image télé, c'est-à-dire cinquante fois par seconde. Quand le ZX81 fonctionne en mode lent, l'image est constamment "rafraîchie" ; la variable FRAMES est donc continuellement sollicitée, ce qui représente 50 "tic-tac" par seconde.

Ce tic-tac étant beaucoup trop rapide pour être contrôlé à l'aide d'un programme Basic, nous avons utilisé l'octet le plus significatif de la variable FRAMES. Ce dernier, situé à l'adresse 16437, est modifié tous les 256 "rafraîchissements" d'écran.

Les données sont réunies pour effectuer le calcul de la base de temps du programme : il faut 256 "rafraîchissements" pour modifier la valeur de l'octet à raison de 50 par seconde. Le résultat de l'opération 256/50 est égal à 5.12 secondes.

Cette valeur est représentative de "l'horloge interne". Si un programme contrôle le contenu de l'adresse 16437, un laps de temps égal à 5.12 secondes s'écoulera chaque fois que le contenu sera modifié. L'horloge fonctionnant en continu, cela n'a aucune incidence si le programme met une ou deux secondes pour exécuter un autre travail.

Tant que l'exécution du programme prend moins de 5.12 secondes avant de contrôler l'horloge, l'heure reste exacte.

Ce programme "Horloge numérique" s'inspire des idées précédentes pour afficher l'heure de façon permanente. La base de temps (égale à 5.12 s.) ne permet pas l'affichage d'une valeur inférieure ; par conséquent, le temps écoulé ne peut être compté seconde par seconde.

Vous pouvez constater de temps à autre un saut de six secondes à la place de celui de cinq secondes. Ceci se produit chaque fois que les ajouts successifs de la partie décimale de la base de temps (soit 0.12) provoquent une retenue (tous les 9 tic-tac).

Maintenant, vous concevez aisément comment la méthode de réglage de cette horloge peut devenir précise. Puisque le programme ne dépend pas de l'instruction PAUSE, n'importe quelle valeur peut être ajoutée chaque fois qu'un tic-tac est détecté. Bien que la valeur 5.12 donnée comme base de temps soit très proche de la vérité, vous constaterez sans doute que l'horloge n'est pas aussi précise que ça. Le programme a été conçu pour que vous puissiez vous-même rectifier cette valeur à la ligne 5.

Faites tourner le programme pendant un délai d'une heure (en chronométrant aussi précisément que possible) et calculez le nombre n de tic-tac émis par l'horloge interne. Vous pouvez, si vous le préférez, modifier le programme pour que l'ordinateur calcule ce nombre à votre place. Calculez ensuite la durée réelle de chaque tic-tac (3600/n) et entrez cette valeur à la ligne 5 du programme. Personnellement, nous avons obtenu une base de temps de 5 secondes.

Le programme "Machine à sous" utilise ce programme pour chronométrer la durée du jeu.

Remarquez que ce programme occupe presqu'entièrement la mémoire d'un ZX81 1K.

```
 1 REM HORLOGE
 5 LET DUREE TIC=5.12            (voir ci-dessus
10 PRINT "HEURE (HHMM)?"         (temps initial
20 INPUT T$
30 LET H=VAL T$(1 TO 2)          (extrait l'heure
40 LET M=VAL T$(3 TO 4)          (extrait les minutes
```

```
  50 LET S=(- DUREE TIC)                    (initialisation du chrono
  60 LET T1=-1                              (contrôle du tic-tac
  70 POKE 16436,255                         (démarre le tic-tac
  80 CLS
  90 PRINT AT 10,10;"***********";          (pour l'esthétique !
           AT 12,10;"***********"
 100 LET T2=PEEK 16437                      (contrôle FRAMES
 110 LET TIC =(T1<>T2)                      (valeur modifiée ?
 120 IF TIC  THEN GOSUB 1000                (oui ?
 130 LET T1=T2                              (réactualise le compteur
 140 GOTO 100                               (nouveau contrôle
1000 LET S=S+DUREE TIC                      (incrémente le temps
1010 IF S<60 THEN GOTO 1100                 (1mn de passée ?
1020 LET M=M+1                              (réactualise les minutes
1030 LET S=S-60                             (réinitialise les secondes
1040 IF M<60 THEN GOTO 1100                 (1 heure de passée ?
1050 LET H=H+1                              (réactualise les heures
1060 LET M=M-60                             (réinitialise les minutes
1070 IF H>23 THEN LET H=0                   (un jour de passé ?
1100 PRINT AT 11,12;H;":";M;":";INT S;"   "  (imprime l'heure
1110 RETURN
```

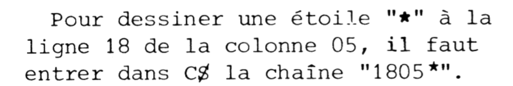
## COMMENT DEVENIR ARTISTE

### *CAPACITE MEMOIRE NECESSAIRE : 1K RAM*

Ce minuscule programme va-t-il éveiller en vous un nouveau Picasso ? D'accord, la comparaison est peut-être un peu exagérée mais il vous permettra tout de même de donner libre cours à votre pensée créatrice sur l'écran.

Le programme demande l'entrée d'une série de "coups de pinceau". Ceux-ci sont représentés par une chaîne de cinq caractères : les deux premiers définissent le numéro de ligne, les deux suivants le numéro de colonne et le dernier le caractère à peindre sur l'écran (à la position déterminée par les quatre premiers chiffres).





Pour dessiner une étoile "*" à la ligne 18 de la colonne 05, il faut entrer dans C$ la chaîne "1805*".

Si vous donniez par hasard un coup de pinceau malheureux, le remède est simple : dessinez à la même position un "espace" (si vous désiriez un blanc) ou un nouveau caractère.

```
10 INPUT C$                                  (entrée des "coups de pinceau"
20 IF LEN C$<>5 THEN GOTO 10                 (vérifie si tout est O.K.
30 PRINT AT VAL C$(1 TO 2),VAL C$(3 TO 4);C$(5);
                                             ("peint"
40 GOTO 10                                   (recommence...
```

Pour vous donner un aperçu, voici un exemple d'application de quelques "coups de pinceau" : les caractères :: sont obtenus en tapant GRAPHICS/SHIFT/A :

```
0920/     1314-     1224::
0921I     1315-     1225::
1019/     1316-     1226::
1021I     1317-     1227::
1114-     1318-     1228::
1115-     1319-     1229::
1116-     1320-     1230::
```

```
1117-        1321-        1121I
1118-        1311-        1221I
1213/        1312-
1214.        1222::
1313-        1223::
```

Les possesseurs de l'extension 16K RAM préfèreront peut-être la variante présentée ci-dessous. Elle a le mérite supplémentaire de permettre la sauvegarde de vos chefs-d'oeuvre sur cassettes. Vous pouvez ainsi les redessiner à volonté !

Lancez l'exécution du programme et introduisez vos "coups de pinceau" comme précédemment (l'exemple ci-dessus convient parfaitement dans ce cas). Quand vous êtes satisfait de votre "peinture", entrez dans C$ la commande SAVE à la place de la chaîne de caractères.
Le programme vous demandera alors de baptiser votre oeuvre pour la sauvegarder sous son nom (s'assurer que le magnétocassette est en route avant d'entrer le nom).
Quand vous chargerez à nouveau le programme, votre dessin sera automatiquement affiché.

A tout moment il est possible d'entrer le mot-clé "QUIT" pour interrompre l'exécution du programme.

```
  1 REM 16K ARTISTE
  2 REM ***********
 10 DIM P$(22,32)                        (copie du dessin
 20 REM DESSIN
 30 CLS
 40 FOR Y=1 TO 22
 50 PRINT P$(Y);
 60 NEXT Y
100 REM COUPS DE PINCEAU
110 INPUT C$
120 IF C$="SAVE" THEN GOTO 1000
130 IF C$="QUIT" THEN GOTO 9999
140 IF LEN C$<>5 THEN GOTO 100           (vérifie la conformité
150 LET Y=VAL C$(1 TO 2)                 (numéro de ligne
160 LET X=VAL C$(3 TO 4)                 (numéro de colonne
170 IF INT (Y/22)<>0 OR INT (X/44)<>0 THEN GOTO 100
180 PRINT AT Y,X;C$(5);                  ("peint" le caractère
```

```
 190 LET P$(Y+1,X+1)=C$(5)              (stockage dans un tableau
 200 GOTO 100                           (recommence
1000 REM SAUVEGARDE DU DESSIN
1010 PRINT AT 21,0;"NOM DU DESSIN?"
1020 INPUT N$
1030 IF LEN N$=0 THEN GOTO 1000         (attention à l'erreur F
1040 SAVE N$
1050 GOTO 20                            (dessine à nouveau l'oeuvre
9999 STOP
```

## *** RECREATION ***

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***

Le ZX81 choisit au hasard un nombre compris entre 1 et 99. A vous de deviner lequel avant que l'ordinateur se lasse de votre inefficacité et ne vous donne la solution.

```
  10 LET N=INT (RND*99)+1
  20 SCROLL
  30 PRINT "JE PENSE A UN NOMBRE..."
  40 FOR G=1 TO 5+INT (RND*3)
  50 SCROLL
  60 PRINT "ENTREZ LE NUMERO ESTIME";G;": ";
  70 INPUT Y
  80 PRINT Y
  90 IF Y=N THEN GOTO 200
 100 SCROLL
 110 PRINT "PAS DU TOUT ";
 120 IF Y>N THEN PRINT "TROP HAUT"
 130 IF Y<N THEN PRINT "TROP BAS"
 140 NEXT G
 150 SCROLL
 160 PRINT "RATE. LE NOMBRE ETAIT ";N
 170 GOTO 9999
 200 SCROLL
 210 PRINT "TROUVE. BRAVO..."
9999 STOP
```

# JUSTIFICATION DÉCIMALE

Un des problèmes posés par le mode de calcul à virgule flottante concerne le formatage des résultats au moment de leur impression.
Sauf avis contraire, le ZX81 ne formate pas les variables numériques ; il les affiche au contraire sous forme de série de chiffres dont la présentation n'est pas toujours adéquate. Il est alors difficile d'afficher des tableaux de résultats où les nombres sont tous bien alignés.

## *FORMATAGE DES PRIX*

Le problème mentionné ci-dessus est particulièrement évident quand un programme manipule des valeurs monétaires.
Un prix est généralement donné avec deux chiffres après la virgule, même s'il s'agit de deux zéros. Imaginons par exemple un programme de calcul de la T.V.A. devant afficher les résultats selon la présentation suivante :

| MONTANT | TAUX T.V.A. | T.V.A. |
|---|---|---|
| 100.00 | 17.6 | 17.60 |
| 1.00 | 17.6 | 0.18 |
| 25.60 | 17.6 | 4.51 |
| .... | .... | .... |
| .... | .... | .... |

Si aucune précaution de formatage n'est prise, lors de l'exécution du programme l'alignement des nombres sera effectué sur le chiffre de gauche : cette opération est appelée *justification à gauche* :

| MONTANT | TAUX T.V.A. | T.V.A. |
|---|---|---|
| 100 | 17.6 | 17.6 |
| 1 | 17.6 | 0.18 |
| 25.6 | 17.6 | 4.51 |
| .... | .... | .... |
| .... | .... | .... |

Le sous-programme présenté ci-dessous reconstitue les nombres suivant un format fixé et les aligne dans une colonne déterminée. On obtient ainsi un affichage des résultats beaucoup plus lisible. Si vous utilisez une imprimante, l'effet sera encore meilleur.

Tous les résultats sont arrondis à deux chiffres après la virgule ; ainsi, une valeur égale à 3.245 devient 3.25 sur l'écran. Soyez néanmoins prudent : le système d'arrondi peut faire apparaître des erreurs de quelques centimes dans le résultat ici ou là.

Essayez au maximum d'utiliser des valeurs à deux décimales pour éviter cet ennui.

Les deux variables suivantes doivent être définies avant l'appel du sous-programme :

- **V** qui contient la valeur à afficher
- **C** qui contient le numéro de la colonne où sera aligné le chiffre de droite de la valeur (justification à droite).

```
9500 REM JUSTIFIE V A C (2 DECIMALES)
9510 LET XL=INT (ABS V+.005)*SGN V          (arrondit les francs
9520 LET XP=INT ((ABS (V-XL)*100)+0.5)      (arrondit les centimes
9530 LET Z$=STR$ XP                         (convertit les centimes
9540 LET Z$=STR$ XL +"."+("0"+Z$)(LEN Z$ TO )
                                            (met des zéros si pas de
                                                          centimes
9550 PRINT TAB (C-LEN Z$+1);Z$;             (affiche à la colonne C
9560 RETURN                                 (terminé
```

Voici un petit exemple d'application de ce sous-programme :

```
100 REM CALCUL DE TVA
110 PRINT "ENTRER LE MONTANT FFF.CC"
120 INPUT V
130 LET S=V                   (stocke le montant pour plus
                                                      tard
140 LET C=18                  (affiche à la colonne 18
150 GOSUB 9500                (justifie V et l'affiche
160 PRINT                     (nouvelle ligne
170 LET V=V*17.6/100          (calcule la T.V.A.
180 GOSUB 9500                (justifie et affiche
190 PRINT                     (nouvelle ligne
200 PRINT TAB 13;"------"     (pour l'esthétique
```

```
210 LET V=V+S                    (valeur + T.V.A.
220 GOSUB 9500                   (justifie et affiche montant total
230 PAUSE 4E4
240 CLS
250 RUN
```

Si vous avez un ZX80, ajoutez : 235 POKE 16437,255.

***JUSTIFICATION AMOVIBLE***

Voici une extension du programme précédent qui vous permet (en plus) de choisir le nombre de décimales dans l'affichage du résultat. En plus des variables **V** et **C**, il faut définir une variable **N** qui contiendra le nombre de décimales désiré.

Ce sous-programme n'effectue pas l'arrondi automatique car dans de nombreux cas le ZX81 n'offre pas une précision suffisante pour les opérations de division (voir chapitre "trucs et astuces").

Mieux vaut donc arrondir vous-même la variable **V** pour être sûr que le degré de précision est suffisant avant d'appeler le sous-programme.

```
9500 REM JUSTIFIE V A C AVEC N DECIMALES
9510 LET Z$=""
9515 FOR Z=1 TO N
9520 LET Z$=Z$+"0"
9525 NEXT Z
9530 LET XL=INT ABS V*SGN V
9535 LET XP=INT (ABS (V-XL)*10**N)
9540 LET Z$=STR$ XL+"."+(Z$(1 TO N-LEN STR$ XP)+
        (STR$ XP+Z$))(1 TO N)
9545 PRINT TAB (C-LEN Z$+1);Z$;
9550 RETURN
```

Ce sous-programme peut être appliqué aux calculs de T.V.A. présentés plus haut à une seule modification près : ajoutez au programme initial une nouvelle ligne permettant d'initialiser la variable **N** à 2 (représente le nombre de chiffres après la virgule).

```
105 LET N=2
```

Ce sous-programme est sensiblement plus lent que le précédent car la séparation "amovible" de la partie décimale, qui constitue tout son intérêt, est effectuée après une élévation à la puissance (**). Il présente par contre l'avantage d'être plus souple.
Si vous ne désirez que deux chiffres après la virgule, contentez-vous de la première version. Sinon, utilisez la deuxième. La variable **N** peut être affectée une fois pour toute ou bien modifiée suivant les besoins de la cause.

## SIMULATION DU PRINT USING

Bien qu'assez proche du Basic Microsoft (qui sert pratiquement de référence pour la plupart des ordinateurs individuels), le Basic du ZX81 présente quelques lacunes par rapport à celui-ci.
Une de ces lacunes concerne l'instruction PRINT, qui ne permet pas véritablement de formatage. Le sous-programme présenté ci-après est une nouvelle extension des deux précédents. Son utilisation permet un formatage numérique encore plus souple.

Un masque représentatif du formatage désiré pour afficher un nombre est affecté à la variable alphanumérique **U$**.
La ligne :

```
LET U$="9999.99"
```

limitera obligatoirement l'affichage du contenu de la variable **V** à quatre chiffres avant la virgule et deux après.
Le chiffre "9" symbolise la position des chiffres dans l'expression du résultat. Le point représente la position relative de la virgule (le point décimal correspond à la notation anglo-saxonne de la virgule). Le Basic du ZX81 ne fournissant que neuf chiffres et demi significatifs, il est inutile d'utiliser un masque comportant plus de neuf (voire dix) caractères.

Ce sous-programme fournit également la possibilité d'affecter à U$ un masque ne comportant aucun chiffre après la virgule, ce qui le rend utilisable pour toutes les valeurs numériques (dans la limite des 9,5 chiffres autorisés ; au-delà le ZX81 passe en notation scientifique, de quoi laisser le sous-programme perplexe !).
Attention, si le résultat nécessite plus de chiffres que n'en définit le masque, le chiffre de gauche (c'est-à-dire le plus significatif) est tronqué.
Comme pour le sous-programme précédent, l'arrondi automatique n'est pas effectué. La variable **V** doit avoir la précision requise avant l'appel du sous-programme.
La variable **U$** peut être affectée une fois pour toute au début du programme ou bien modifiée en cours de programme si besoin est.

Les trois variables requises sont les suivantes :

**V** qui contient la valeur à afficher
**C** qui contient le numéro de colonne sur laquelle est aligné le chiffre de droite
**U$** qui contient le masque désiré (voir ci-dessus)

```
9500 REM PRINT USING U$
9505 LET Z$=""                                  (initialise le masque
9510 LET XL=0                                   (pas de point décimal trouvé
9515 FOR Z=1 TO LEN U$                          (cherche le point décimal
9520 IF XL THEN LET Z$=Z$+"0"                   (réactualise Z$ si le point
                                                         décimal trouvé
9525 IF U$(Z)<>"." THEN GOTO 9535               (regarde si le masque a un"."
9530 LET XL=NOT XL                              (met à 1 si point décimal
                                                               trouvé
9535 NEXT Z
9540 LET XL=INT ABS V*SGN V                     (comme précédemment
9545 LET XP=INT (ABS (XL-V)*10**LEN Z$)
9550 IF LEN Z$ THEN GOTO 9565                   (vérifie s'il y a des décimales
9555 LET Z$=STR$ XL                             (sinon affiche la partie entièr
9560 GOTO 9570
9565 LET Z$=STR$ XL+"."+(Z$(1 TO LEN Z$-LEN STR$ XP)+
        STR$ XP+Z$)(1 TO LEN Z$)
9570 IF LEN Z$>LEN U$ THEN LET Z$=Z$(LEN Z$-LEN U$+1 TO )
                                                (tronque si nécessaire
9575 PRINT TAB (C-LEN Z$+1);Z$;
9580 RETURN
```

## ECART-TYPE

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***

Ce petit programme permet de calculer l'écart-type d'une série de données. Pour clore la liste de données, il suffit d'introduire le mot **FIN** au lieu de la nouvelle valeur demandée.

Pour éviter toute erreur d'introduction, l'ordinateur vous indique également le numéro d'ordre de la valeur à entrer.



Si vous vous interrogez pour savoir comment il est possible d'entrer une variable numérique à la place d'une valeur numérique, reportez-vous au chapitre "trucs et astuces".

```
   1 REM ECART TYPE
  10 LET POINTS=0                          (initialisation des variables
  20 LET  CARRES=0
  30 LET SOM=0
  40 LET FIN=1E38                          (modification éventuelle
 100 CLS                                   (boucle principale
 110 PRINT "ENTREZ LA DONNEE";POINTS+1     (demande la donnée
 120 INPUT VALEUR                          (entre cette donnée
 130 IF VALEUR=FIN THEN GOTO 200           (est-ce fini ?
 140 LET POINTS=POINTS+1                   (actualise le numéro d'entrée
 150 LET SOM=SOM+VALEUR                    (actualise la somme
 160 LET CARRES =CARRES+VALEUR**2          (somme des carrés
 170 GOTO 100                              (valeur suivante
 200 CLS                                   (affiche les résultats
 210 PRINT "ECART TYPE: ";
 220 PRINT SQR ( CARRES/POINTS-(SOM/POINTS)**2)
9999 STOP
```

## LES DENTS DE LA MER

***CAPACITE MEMOIRE NECESSAIRE : 16K RAM***

Vous vous sentez des envies de mordre ? Alors ce programme va vous donner satisfaction ; il vous transforme en requin affamé poursuivant et terrorisant quatre innocents baigneurs.

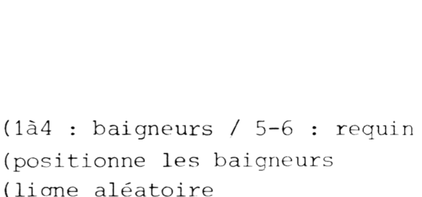
Essayez de les "croquer" tous les quatre en un minimum de temps.

Faites avancer le requin en appuyant sur les touches 5, 6, 7 et 8 (sans utiliser la touche SHIFT). Celui-ci se déplace dans le sens de la flèche correspondant à la touche pressée.

Une fois la direction modifiée, il est inutile de continuer à appuyer sur la touche : le requin continue tout seul.

Les quatre baigneurs, dont la position est déterminée aléatoirement, se déplacent sur des axes différents. Leur vitesse de déplacement est égale au quart de celle du requin.

Quand un nageur atteint le bord de l'écran, il "fait le tour" et réapparaît de l'autre côté.



```
  1 REM LES DENTS DE LA MER
  2 REM *******************
 10 DIM S(6,2)                  (1à4 : baigneurs / 5-6 : requin
 20 FOR X=1 TO 4                (positionne les baigneurs
 30 LET S(X,1)=INT (RND*64)     (ligne aléatoire
 40 LET S(X,2)=INT (RND*44)     (colonne aléatoire
 50 NEXT X
 60 LET S(5,1)=30               (positionne le requin
 70 LET S(5,2)=24
 80 LET YD=4                    (initialise la direction du requin
 90 LET T=0                     (nombre de coups
100 LET N=1                     (baigneur suivant
110 PRINT AT 10,14;"DENTS"
200 REM BOUCLE PRINCIPALE
```

```
210 LET X=1                           (initialise compteur de baigneurs
                                                               croqués
220 FOR Y=1 TO 4                      (contrôle chaque baigneur
230 IF S(Y,1)<>S(5,1) OR S(Y,2)<>S(5,2) THEN GOTO 300
                                      (regarde si le requin est après
                                                          le baigneur
240 PRINT AT 21-INT (S(Y,2)/2),INT (S(Y,1)/2);"MIAM"
                                      (oh !  le pauvre...
250 PAUSE 100
260 LET S(Y,1)=-1                     (porte le baigneur disparu
300 IF S(Y,1)<>-1 THEN LET X=0        (reste-t-il des baigneurs vivants ?
310 NEXT Y
320 IF NOT X THEN GOTO 500            (si oui, continue la poursuite
330 PRINT AT 20,0;"REUSSI EN ";T;" COUPS"
340 GOTO 9999
500 LET D=CODE INKEY$ -32             (changement de direction
510 IF D>0 AND D<5 THEN LET YD=D      (vérifie la validité
520 LET D=YD                          (déplace le requin
530 GOSUB 1000                        (calcule la direction du requin
540 UNPLOT S(6,1),S(6,2)              (efface la queue du requin
550 LET S(6,1)=S(5,1)                 (la queue vient à la place de la
560 LET S(6,2)=S(5,2)                                             tête
565 LET S=5                           (indice du requin
570 GOSUB 1100                        (dessine la nouvelle tête
580 LET S=N                           (contrôle le baigneur suivant
590 IF S(S,1)=-1 THEN GOTO 900        (est-il avalé ?
600 LET D=N                           (déplace le baigneur
610 GOSUB 1000                        (vérifie sa direction
620 UNPLOT S(S,1),S(S,2)              (efface l'ancienne position
630 GOSUB 1100                        (dessine la nouvelle
900 LET T=T+1                         (incrémente le compteur de coups
910 LET N=N+1                         (réactualise le nombre de baigneurs
920 IF N>4 THEN LET N=1               (seulement quatre
930 GOTO 200                          (continue...
1000 REM CALCULE LA DIRECTION
1010 LET DX=((D=1)*-1)+(D=4)          (DX contient une seule position
1020 LET DY=((D=2)*-1)+(D=3)          (DY contient une seule position
1030 RETURN
1100 REM GESTION DE MOUVEMENTS
1110 LET S(S,1)=S(S,1)+DX             (déplace d'une position
1120 LET S(S,2)=S(S,2)+DY             (déplace d'une position
```

```
1130 LET S(S,1)=S(S,1)-INT (S(S,1)/64)*64 (contrôle "tour par derrière"
1140 LET S(S,2)=S(S,2)-INT (S(S,2)/44)*44 (contrôle "tour par derrière"
1150 PLOT S(S,1),S(S,2)                   (dessine nouvelle position
1160 RETURN
```

Si vous utilisez un ZX80, ajoutez les lignes suivantes :

```
505 PAUSE 25
506 POKE 16437,255
```

## *** RECREATION ***

Vous pourrez exécuter ce minuscule programme à l'un de vos moments perdus ; il dessine de lui-même des esquisses quelquefois amusantes. Désolé pour les utilisateurs de ZX80, celui-ci ne fonctionne qu'en mode lent.

Vous pouvez toujours essayer de l'adapter mais vous trouverez probablement les sauts d'image fatigants à la longue.

```
10 LET P=0
20 LET X=INT (RND*32)+16
30 LET Y=INT (RND*22)+11
40 IF P THEN PLOT X,Y
50 IF NOT P THEN UNPLOT X,Y
60 LET P=NOT P
70 GOTO 20
```

# UTILISATION DES CODES MACHINES

Ce chapitre n'a pas la prétention de vous apprendre à programmer en langage machine.

Son contenu se limite à l'étude des méthodes utilisées pour entrer un sous-programme en langage machine. Ces méthodes sont au nombre de trois et obéissent à des règles très précises ; comme vous le constaterez, chacune présente des avantages et des inconvénients.

La méthode choisie dépendra largement du nombre de codes machine à entrer et de la manière dont ils sont écrits.

## UTILISATION DE L'INSTRUCTION REM

Cette méthode est certainement la plus commode pour entrer un sous-programme et le sauvegarder sur cassette avec le programme principal.

La première ligne du programme doit comporter une instruction REM ; celle-ci est suivie d'une série de caractères "quelconques" qui sont appelés à être remplacés par le sous-programme.

La première ligne du programme est placée à l'adresse 16509 de la mémoire. L'adresse du premier caractère "quelconque" est 16514 (16509+2 octets pour le numéro de ligne+2 octets pour la longueur de la ligne+un octet représentant l'instruction REM).

Ceci peut être vérifié en tapant :

```
NEW
10 REM ABCDEF
PRINT CHR$ (PEEK 16514)
```

ce qui a pour conséquence l'affichage de la lettre "A".

Grâce à l'instruction POKE, des sous-programmes en code machine peuvent être entrés à partir de cette adresse. Chaque code entré

viendra remplacer un des caractères "quelconques" suivant l'instruction REM.

Ne vous inquiétez pas si, après cette opération, la commande LIST vous donne une ligne d'instruction REM contenant des signes bizarres ; le ZX81 essaye de convertir en caractères tous les codes machine introduits et les affiche à tort.
En voici un exemple :

```
  1 REM AAAAA
 10 LET X=16514
 20 POKE X,237
 30 POKE X+1,75
 40 POKE X+2,7
 50 POKE X+3,64
 60 POKE X+4,201
100 PRINT USR X
```

Ce petit programme crée une routine en code machine dont la mission consiste à fournir le numéro de la ligne exécutée. D'une façon générale, on obtient le numéro de la ligne en cours d'exécution à chaque fois que l'instruction USR X est rencontrée.

Ce sous-programme offre la possibilité de faire de l'adressage relatif. Il est à présent tout à fait correct d'écrire :

```
120 GOTO USR X+200
```

L'exécution de cette ligne branche le programme à l'adresse relative du numéro de la ligne en cours d'exécution, c'est-à-dire à l'adresse 320 (dans ce cas, USR X donne 120).

Si vous désirez sauvegarder le programme avec cette routine en code machine, n'oubliez pas d'effacer les lignes 20 à 60, devenues inutiles ; cette opération libère de la place pour d'autres lignes de programme.

Le sous-programme n'est pas remis en cause par l'entrée de la commande RUN car les valeurs correspondant aux codes machine sont insérées dans une ligne de programme et non pas affectées à des variables.

***Les inconvénients*** de cette méthode se résument en deux points :

1- L'écran est encombré de signes incohérents quand vous demandez la liste d'instructions du programme.
2- Le sous-programme doit obligatoirement être situé à la première ligne du programme, ligne normalement réservée à l'identification (instruction REM).

***Les avantages*** sont les suivants :

1- Le sous-programme n'est pas affecté par les commandes RUN et CLEAR.

2- Le sous-programme est facilement localisé, ce qui facilite son accès (adressage direct - JUMP - ou appel de sous-programme - CALL).
3- Après son exécution, le sous-programme devient partie intégrante du programme principal ; les lignes ayant servi à le mettre en place peuvent donc être effacées sans problème.

## *UTILISATION D'UN TABLEAU*

La seconde méthode fait appel à la création d'un tableau qui rassemble tous les codes composants le sous-programme.

La commande DIM permet de préciser la dimension d'un tableau à partir du moment où elle est exécutée. Si la commande DIM est placée à la première ligne du programme, le tableau occupera la première position de la mémoire réservée au stockage des variables. Il est possible de connaître l'adresse correspondante en analysant le contenu de la variable système VARS.

En supposant que le tableau rassemble des variables à une seule dimension (chaîne de longueur fixe par exemple), on peut obtenir l'adresse du premier caractère de la chaîne en exécutant la ligne suivante :

```
LET ADRESSE=PEEK 16400+PEEK 16401*256+6
```

La page 174 du manuel d'utilisation Sinclair nous apprend également qu'il faut d'emblée six octets pour créer un tableau à une seule dimension (voir "Tableau de caractères" page 174). Si cette précaution est omise, l'instruction POKE écrit les codes machine là où il ne faut pas.

L'exemple donné dans le paragraphe précédent peut être modifié de la façon suivante :

```
 1 DIM X$(5)
10 LET X=PEEK 16400+PEEK 16401*256+6
```

La suite du programme est identique ; en effet, la variable X utilisée pour localiser la position du sous-programme est indépendante du choix de la méthode (tableau ou instruction REM).

Il est possible d'utiliser l'instruction CHR$ pour affecter les valeurs à l'aide de l'instruction LET. Une variante du programme précédent pourrait être :

```
  1 DIM X$(5)
 10 LET X=PEEK 16400+PEEK 16401*256+6
 20 LET X$(1)=CHR$ 237
 30 LET X$(2)=CHR$ 75
 40 LET X$(3)=CHR$ 7
 50 LET X$(4)=CHR$ 64
 60 LET X$(5)=CHR$ 201
100 PRINT USR X
```

Ici également, les lignes 20 à 60 peuvent être effacées après l'exécution du programme.

***Avantages* :**

1- La première ligne du programme n'est pas obligatoirement une instruction REM factice. En effet, il est possible de placer des instructions (exemple : PRINT, FAST, SLOW...) avant l'instruction DIM tant que celle-ci précède la première affectation de variable.
2- Le sous-programme peut être manipulé à l'aide des commandes habituelles du langage Basic.
3- Le sous-programme peut être sauvegardé avec le programme principal, ce qui permet l'élimination des lignes ayant servi à le mettre en place.

***Inconvénients***

1- Un sous-programme doit être relogeable, ce qui implique obligatoirement l'utilisation des instructions de branchement relatif. Les instructions d'appel à un sous-programme peuvent être simulées en intervenant directement sur la pile ou en utilisant l'adressage relatif.
2- Toutes les variables doivent être stockées avec le programme. Ceci rend impossible d'une part l'utilisation des commandes RUN et CLEAR, d'autre part le "redimensionage" des tableaux.
3- Comme il n'existe que 26 variables distinctes (A, B... Z), l'emploi de cette méthode peut s'avérer très restrictif dans certains cas.

## UTILISATION DES DERNIERES ADRESSES DE LA MEMOIRE

Cette méthode est une des plus fiables. En effet, les codes machine entrés ne sont pas affectés par les commandes du langage Basic (NEW y compris). Le seul moyen d'anéantir le sous-programme est... d'éteindre l'ordinateur !

Malheureusement, le ZX81 est uniquement prévu pour charger (LOAD) ou sauvegarder (SAVE) programmes et variables ; l'ensemble des codes machine placés aux dernières adresses de la mémoire n'est pas affecté par ces deux commandes.

La variable-système RAMTOP contient l'adresse du dernier octet disponible en "haut de mémoire". Cette valeur est déterminée lors de la mise en route du ZX81.

A tout moment l'instruction POKE permet d'abaisser du nombre d'octets nécessaires la valeur de l'adresse affectée à la variable RAMTOP. Il vous est alors possible d'écrire à partir de cette adresse les codes machine du sous-programme ; rien ne vous empêche de les y laisser indéfiniment si le coeur vous en dit.

Un des inconvénients de cette méthode est l'obligation d'entrer la commande NEW pour rendre effective la modification. Sinon, toute tentative pour entrer des codes machine dans la partie de la mémoire située au-delà de l'adresse définie par la variable RAMTOP serait vaine.

Dans ce but, nous avons mis au point un programme particulier de chargement en mémoire ; celui-ci contient tous les codes machine à placer en "haut de mémoire".

Dans ce cas, vous devez dans l'ordre :

- abaisser la valeur de la variable-système RAMTOP du nombre d'octets requis pour l'exécution du sous-programme.
- taper la commande NEW.
- entrer et exécuter le programme de chargement. Celui-ci placera les codes machine à partir de la nouvelle adresse définie par la variable RAMTOP.

Si ce sous-programme est présent au début de chaque cassette, le programme est capable de charger lui-même un autre programme.

Le programme de chargement ayant généralement pour seul objet de placer les codes machine en haut de mémoire, il est appelé à dévenir assez élaboré. Le programme ci-après donne à tout moment le nombre approximatif d'octets encore disponibles en mémoire. Cette estimation n'est jamais très précise étant donné que l'occupation de la mémoire varie au cours de l'exécution du programme ; elle donne tout de même une idée générale.

Entrez en mode direct :

```
LET R=PEEK 16388+PEEK 16389*256 (donne la valeur de RAMTOP
LET R=R-20                      (diminue de 20
POKE 16388,R-256*INT (R/256)    (la remplace
POKE 16389,INT (R/256)
NEW                             (à ne pas oublier !
```

A présent, entrez, sauvegardez puis exécutez le programme suivant :

```
10 REM CHARGEMENT CODE MACHINE
20 LET RAMTOP=PEEK 16388+PEEK 16389*256
30 LET C$="21000039ED5B1C40ED52444DC9"
40 FOR X=1 TO LEN C$-1 STEP 2
50 POKE RAMTOP+INT ((X-1)/2),
     (CODE C$(X)-28)*16+CODE C$(X+1)-28
60 NEXT X
```

Un tel programme vous permet d'entrer les codes hexadécimaux d'un sous-programme en langage machine. L'ensemble des codes hexadécimaux est placé dans la variable **C$** (ligne 30). La partie du programme comprise entre les lignes 40 et 60 extrait les caractères deux par deux, les convertit en nombres décimaux et les entre en mémoire (POKE) à partir de l'adresse définie par la variable RAMTOP.

Bien que ce programme soit employé pour charger une routine calculant le nombre d'octets encore disponibles en mémoire, il peut servir à charger d'autres programmes en code machine. Il suffit pour cela de modifier la chaîne **C$** de la ligne 30.

Une fois chargée, la routine reste en "haut de mémoire" et ceci jusqu'à ce qu'il y ait coupure de l'alimentation du ZX81.

Elle peut être appelée à tout moment en entrant la ligne :

```
PRINT USR (PEEK 16388+PEEK 16389*256)
```

Vous obtiendrez en retour le nombre d'octets restés disponibles dans la mémoire du ZX81. Ceci correspond, dans le diagramme d'organisation à la zone notée "RESERVE" (voir manuel Sinclair page 171).

La correspondance entre instructions (ou mnémonique) en langage assembleur et les codes entrés dans **C$** se présente de la façon suivante :

| instructions assembleur | | valeurs hexadécimales | |
|---|---|---|---|
| [RAMTOP]: | LD HL,0 | 21 00 00 | (initialise HL à 0 |
| | ADD HL,SP | 39 | (obtient le pointeur de pile |
| | LD DE,(STKEND) | ED 5B 1C 40 | (obtient STKEND |
| | SBC HL,DE | ED 52 | (donne pile moins STKEND |
| | LD B,H | 44 | (réponse dans BC |
| | LD C,L | 4D | |
| | RET | C9 | (retour au Basic |

***Avantages***

1- Le sous-programme obtenu n'est pas affecté par les commandes NEW et CLEAR.

2- Les sous-programmes très longs peuvent être entrés grâce à ce programme spécialement étudié à cet effet ; un maximum de place reste ainsi disponible pour d'autres sous-programmes.

***Inconvénients***

1- Cette méthode longue et ennuyeuse oblige à placer le sous-programme en "haut de mémoire", puisque la commande NEW devra bien être utilisée à un moment ou à un autre.

2- A moins d'être très organisé dans votre travail, vous devrez utiliser les instructions d'adressage relatif (JUMP) et de branchement relatif à un sous-programme (CALL). Sans cette précaution, il peut être assez pénible d'ajouter par la suite des sous-programmes supplémentaires.

***GENERALITES***

Nous avons énuméré plus haut une série de commandes directes qui réduisent la valeur de la variable-système RAMTOP de vingt octets.

Le programme suivant mérite de figurer au début de chaque cassette si vous décidez de réserver, lors de l'allumage du ZX81, la place des routines en code machine.

A chaque fois que vous désirez charger un sous-programme en code machine (en utilisant par exemple le programme ci-dessus), exécutez en premier lieu le programme suivant ; il permet de réserver la place nécessaire.

```
 1 REM INITIALISATION RAMTOP
 2 REM
 3 REM
10 PRINT "COMBIEN D OCTETS A RESERVER?"
20 INPUT N
30 LET R=PEEK 16388+PEEK 16389*256 (donne la valeur actuelle de
40 LET R=R-N                       (calcule nouvelle valeurRAMTOP
50 POKE 16388,R-256*INT (R/256)    (la remet en mémoire
60 POKE 16389,INT (R/256)          (la remet en mémoire
70 NEW                             (réinitialise la mémoire
```

Vous pouvez à présent exécuter le programme de chargement directement.

Si vous avez une prédilection pour les codes machine, vous trouverez le programme suivant particulièrement utile. Il permet d'examiner le contenu d'une portion donnée de la mémoire, tout en spécifiant les adresses en système décimal ou hexadécimal. Il reconnaît même certaines variables-système comme pointeurs d'adresse de la zone mémoire à examiner.

En éliminant le sous-programme démarrant à la ligne 2000, il est possible de tout caser dans une mémoire 1K RAM. Malheureusement, le programme perd dans ce cas une grande part de son efficacité.

```
100 REM AFFICHAGE CONTENU MEMOIRE
110 GOSUB 9000                   (affiche les règles
120 LET A=0                      (adresse mémoire
200 GOSUB 1000                   (demande adresse de départ
210 CLS
220 FOR Y=0 TO SL                (nombre de lignes à afficher
230 PRINT N+Y*8;TAB 7;           (adresse de départ
240 FOR X=0 TO 7                 (affiche 8 octets
250 LET Z=PEEK (N+Y*8+X)         (donne le contenu de l'octet
260 PRINT CHR$ (INT (Z/16)+28);  (convertit en hexadécimal
270 PRINT CHR$ (Z-(INT (Z/16)*16)+28);
280 PRINT " ";
290 NEXT X                       (octet suivant
300 PRINT
310 NEXT Y                       (ligne suivante sur l'écran
320 LET A=N+(SL+1)*8             (réactualise l'adresse implicite
330 PRINT
```

```
 340 GOTO 200                        (affiche sur l'écran
1000 REM ENTREE DE L ADRESSE
1010 PRINT "ENTREZ L ADRESSE (N=CONTINUE)"
1020 INPUT S$
1030 IF NOT LEN S$ THEN GOTO 1020  (donnée entrée dans S$ ?
1040 IF S$(LEN S$)="H" THEN GOTO 1100 (vérifie si la valeur est hexad
1050 GOSUB 2000                     (vérifie noms des variables-
                                                          système
1060 IF N THEN RETURN               (retour si nom particulier
1070 LET N=VAL S$                   (sinon, c'est en décimal
1080 RETURN
1100 REM CONVERSION DE L HEXADECIMAL EN DECIMAL
1110 LET N=0
1120 FOR X=1 TO LEN S$-1
1130 LET N=N*16+CODE S$(X)-28
1140 NEXT X
1150 RETURN
2000 REM CONTROLE ADRESSES PARTICULIERES
2010 LET SL=16                      (limite de l'écran
2020 LET N=0                        (adresse convertie
2030 IF S$="N" THEN LET N=A
2040 IF S$="PROG" THEN LET N=16509
2050 IF S$="VARS" THEN GOTO 2100
2060 IF S$="DFILE" THEN GOTO 2120
2070 IF S$="RAMTOP" THEN GOTO 2140
2080 RETURN
2100 LET N=PEEK 16400+PEEK 16401*256
2110 RETURN
2120 LET N=PEEK 16396+PEEK 16397*256
2130 RETURN
2140 LET N=PEEK 16388+PEEK 16389*256
2150 RETURN
9000 REM INSTRUCTIONS
9010 PRINT TAB 8;"AFFICHE MEMOIRE EN HEX"
9020 PRINT
9030 PRINT "ENTREZ L ADRESSE DE DEPART :  ",
           "(1) EN VALEUR DECIMALE",
           "(2) EN HEXADECIMAL (EX 4EA3H)",
           "(3) ""N"" (SIGNIFIE ""CONTINUE"")",
           "(4) OU LE NOM D UNE VAR.SYSTEME-",
           "    PROG,VARS,DFILE,RAMTOP"
9040 PRINT
9050 PRINT "(PRESSEZ UNE TOUCHE)"
```

```
9060 PAUSE 4E4
9070 POKE 16437,255
9080 CLS
9090 RETURN
```

En l'absence d'extension 16K RAM, remplacer les lignes 2000 à 2150 par :

```
2000 LET SL=8
2010 LET N=0
2020 RETURN
```

Eliminez également les instructions REM, puis effacez les lignes 1040, 1100 à 1150, 9000 à 9090 et enfin la ligne 110. Vous pouvez alors entrer les valeurs en système décimal mais vous perdez la possibilité d'entrer le nom des variables-système (désolé).

## ***COMBINAISON DES SOUS-PROGRAMMES***

Ce sous-programme en code machine - très sophistiqué - permet de combiner différentes routines entre elles. Il permet également de spécifier quelle fonction particulière est requise à un moment donné.

Les variables-système contiennent une zone réservée de deux octets à l'adresse 16507. Celle-ci n'est pas affectée par le système Basic, bien qu'elle soit effacée chaque fois que la commande NEW est introduite.

Le sous-programme utilise cette zone pour spécifier quelle est la fonction requise ; en introduisant un nombre à cette adresse, il effectue le choix approprié. **L'appendice B** fournit un listing de ce sous-programme en langage assembleur, bien que les principales caractéristiques soient développées dans ce chapitre.

Le sous-programme doit obligatoirement commencer au début d'une page-mémoire, ce qui implique que l'adresse doit être un multiple de 256. A chaque fois que vous allumez votre ZX81, vous remarquerez que la valeur de la variable RAMTOP est un multiple de 256. En exécutant le programme d'initialisation donné plus haut, vous pouvez réserver 256 octets et être certain de situer la routine au début d'une page-mémoire.

Si vous désirez plus tard ajouter d'autres routines, analysez attentivement les listings fournis dans l'appendice.

Comme indiqué, les fonctions de la routine sont :

| Valeur écrite à l'adresse 16507 | Valeur rendue par USR |
|---|---|
| 0 | Estimation de la mémoire restante |
| 1 | Adresse du fichier écran |
| 2 | Adresse des variables du Basic |
| 3 | Adresse de la mémoire libre au-dessus de la routine elle-même |
| 4 | Numéro de ligne en cours |

La plupart de ces fonctions pour être assez banales, n'en sont pas moins extrêmement utiles ; elles évitent de recourir à une ligne additionnant les deux octets obtenus par l'instruction PEEK.

Voici un exemple d'application de ce sous-programme :

```
 20 LET R=PEEK 16388+PEEK 16389*256 (donne l'adresse de la routine
 30 LET S=16507                     (adresse variable-système libre
100 POKE S,0                        (estimation de la mémoire
                                                     restante
110 LET RESTE=USR R                 (exécution
120 PRINT "IL Y A ";RESTE;" OCTETS LIBRES"    (affichage
130 POKE S,2                        (adresse des variables
140 LET V=USR R                     (V contient maintenant l'adresse
                                    (des variables du Basic

....
....
```

Les lignes 130 et 140 remplacent en fait :

```
130 LET V=PEEK 16400+PEEK 16401*256
```

et vous économisent 18 octets en mémoire.

Que peut-on encore dire de ce super sous-programme ? Vous pouvez l'entrer en mémoire en modifiant la ligne 30 du programme "Chargement des codes machine" de la façon suivante :

```
30 LET C$="3A7B40FE05D05F1600211
00060196EE91522272C3121000039ED5
B1C40ED52444DC9ED4B0C40C9ED4B104
0C901003644C9ED4B0740C9"
```

(La présentation correspond à l'affichage obtenu sur l'écran au moment de l'introduction).

Lancez l'exécution du programme de chargement puis essayez le programme de test présenté plus haut pour déterminer le nombre d'octets restant disponibles. Quand on entre un programme particulièrement important (à plus forte raison si l'extension 16K est

reliée), il est toujours bon de savoir si l'on court à la catastrophe ou non. Un programme long est suffisamment ennuyeux à taper : mieux vaut savoir le plus tôt possible si l'on n'est pas trop "juste" en mémoire.

## *** RECREATION ***

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***

Essayez d'échapper à votre adversaire (le ZX81). Après un court instant (déterminé aléatoirement), l'ordinateur ouvre le feu sur vous. En appuyant sur une touche avant que le ZX81 n'ait eu le temps de tirer, vous pouvez avoir la vie sauve.

Ne vous précipitez pas trop non plus, sinon il tirera quand même !

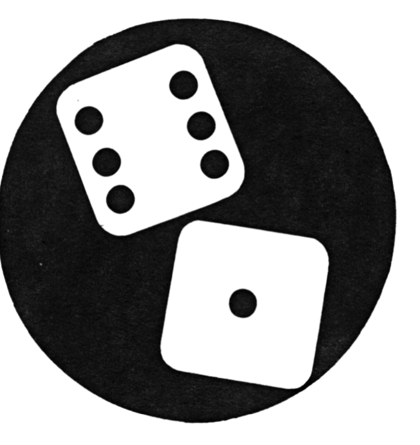
```
 10 LET R=INT (RND*200)
 20 FOR X=1 TO R+50
 30 IF INKEY$<>"" THEN GOTO 100
 40 NEXT X
 50 PRINT AT 11,14;"**FEU**"     (utilisation des caractères inverses
 60 GOTO 200
100 IF R-X<10 THEN GOTO 150
110 PRINT AT 9,15;"EN JOUE"
120 FOR X=X TO R
130 NEXT X
140 GOTO 50
150 PRINT AT 11,14;"CLIC...",,,,"SAUVE"
200 PAUSE 4E4
210 CLS
220 RUN
```

Si vous possédez un ZX80, modifiez les lignes 10 et 100 de la façon suivante :

```
 10 LET R=INT (RND*800)
100 IF R-X<40 THEN GOTO 150
```

## JEU DE DES

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***



Comment épater vos amis ? Si vous appréciez particulièrement les jeux de société, alors ne manquez surtout pas ce programme. Il jette deux dès et vous montre même les faces qui sont sorties ! Après chaque tour, il suffit d'appuyer sur une touche pour jeter à nouveau les dés.

Qui croirait qu'un ordinateur d'à peine 1000 F est capable de simuler une partie de dés ? C'est pourtant ce qu'il fait.

Il a en plus un énorme avantage par rapport aux dés que vous connaissez tous : point n'est besoin de se mettre à quatre pattes pour les chercher s'ils ont la fantaisie de rouler sous un meuble !

La technique utilisée pour écrire ce programme présente une innovation par rapport aux précédentes. Les idées qui l'inspirent ont été introduites dans un des chapitres précédents ("Optimisation des programmes"). Nous avons ici une occasion de les voir à l'oeuvre.

Le nombre de valeurs numériques est limité au strict minimum. Parce qu'elles occupent à elles seules six octets dans une ligne de programme, les expressions numériques sont une source de gaspillage quand on les utilise fréquemment.

Les lignes 10 à 30 du programme servent à définir trois variables qui contiennent respectivement les valeurs 2, 4 et 8. Remarquez le mode d'écriture ; même s'il semble un peu lourd, il permet d'occuper moins de place en mémoire que LET T=2, LET F=4 et LET E=8. En effet, la ligne LET F=T+T n'occupe que 11 octets alors que LET F=4 en occuperait 15.

Les variables T, F et E ont été employées partout où les valeurs numériques 2, 4 et 8 étaient nécessaires. Dans certains cas, c'est la valeur 12 qui est voulue : on utilise alors la somme E+F à la place, ce qui économise 5 octets à chaque fois.

Il est particulièrement intéressant de noter que la valeur 1 peut être obtenue en divisant n'importe quelle variable par elle-même ; l'expression T+T non seulement est égale à 1, mais encore occupe

quatre octets de moins en mémoire. Cela peut sembler quelque peu ridicule à certains, mais dans ce cas, la vitesse d'exécution a été délibérément sacrifiée au profit de la place en mémoire.

A titre d'exercice, il vous est possible de remplacer E, F, et T par les valeurs numériques correspondantes. Dans ce cas, vous noterez que lors de l'introduction du programme la mémoire se remplit sérieusement entre les lignes 1100 et 1200.

Le programme n'aura même pas le temps d'être exécuté que le code erreur 4 sera affiché (pour le modèle 1K). L'idée de départ n'est donc pas sans fondements ; aussi, surveillez attentivement la quantité de valeurs numériques que vous utilisez dans un programme.

```
   1 REM JEU DE DES
   2 REM
  10 LET T=2                    (affecte les valeurs numériques à des
                                                            variables
  20 LET F=T+T
  30 LET E=F+F
  40 RAND
 100 CLS
 110 LET A=E                    (A et B sont les positions sur l'écran
 120 LET B=T+F
 130 GOSUB 1000                 (dessine une face de dé
 140 LET A=E*F                  (place le 2ème dé
 150 GOSUB 1000                 (dessine le 2ème dé
 200 PAUSE 4E4                  (pause à l'infini...
 210 POKE 16437,255
 220 GOTO 100                   (continue à faire la même chose
1000 LET S=INT (RND*(T+F))+T/T     (tire un nombre au hasard entre
                                                           1 et 6
1010 FOR X=A TO A+E+E           (dessine les contours
1020 PLOT X,B
1030 PLOT X,B+E+E
1040 NEXT X
1050 FOR X=B TO B+E+E
1060 PLOT A,X
1070 PLOT A+E+E,X
1080 NEXT X
1100 IF S<>INT (S/T)*T THEN PLOT A+E,B+E  (dessine ce qui est sorti
1110 IF S<T THEN RETURN
1120 PLOT A+F,B+E+F
1130 PLOT A+E+F,B+F
1140 IF S<F THEN RETURN
1150 PLOT A+E+F,B+E+F
1160 PLOT A+F,B+F
```

```
1170 IF S<T+F THEN RETURN
1180 PLOT A+F,B+E
1190 PLOT A+E+F,B+E
1200 RETURN
```

## *** RECREATION ***

*Tout à fait adapté à la version 1K RAM*

Le jeu consiste à tirer sur la cible au moment où vous passez devant. Appuyez sur n'importe quelle touche pour tirer ou pour recommencer une partie.

Ce programme est exécuté en mode lent ; les utilisateurs de ZX80 doivent ajouter des instructions PAUSE aux moments opportuns.

```
 10 LET T=INT (RND*40)+4
 20 PLOT 63,T
100 FOR Y=0 TO 43
110 PLOT 0,Y
120 IF INKEY$<>"" THEN GOTO 200
130 NEXT Y
140 PRINT AT 20,8;"TROP TARD"
150 GOTO 300
200 FOR X=0 TO 63
210 PLOT X,Y
220 NEXT X
230 IF Y=T THEN PRINT AT 21,8;"DANS LE MILLE"
240 IF Y<>T THEN PRINT AT 21,8;"RATE"
300 PAUSE 4E4
310 CLS
320 RUN
```

Pour les possesseurs de ZX80, il suffit d'ajouter les lignes :

```
115 PAUSE 10
305 POKE 16437,255
```

# CONVERSION NUMÉRIQUE

***POURQUOI NE PAS UTILISER LA FONCTION VAL ?***

Bien que la fonction **VAL** du ZX81 soit très utile, elle n'est pas tout à fait conforme aux conventions du Basic Microsoft. Cela ne signifie en aucun cas qu'elle soit mauvaise ; elle est simplement différente.

Il y a cependant un problème : il est impossible d'entrer une chaîne de caractères par l'intermédiaire de l'instruction INPUT puis d'utiliser la fonction **VAL** pour savoir si c'est une valeur numérique qui a été entrée. En effet, si la chaîne n'est pas une expression numérique, l'utilisation de la fonction **VAL** affiche le code erreur B.

Pourquoi entrer un nombre sous forme de chaîne de caractères ? Il y a plusieurs raisons, la principale consistant à rendre le programme plus fiable.

En clair, cela signifie qu'une information non conforme sera dépistée avant que le ZX81 n'ait une chance d'afficher un code erreur. Prenons l'exemple suivant :

```
10 PRINT "QUEL AGE AVEZ VOUS? ";
20 INPUT A
30 PRINT A
```

Si vous essayez d'exécuter ce programme en entrant n'importe quoi dans la variable A (par exemple CHIEN ou CHAT... ), vous obtiendrez à coup sûr le code erreur 2/20, voire pire ! Ceci ne vous sera pas d'un grand secours si vous entrez une mauvaise réponse au beau milieu d'un programme.

Le sous-programme présenté ici vous permet de convertir une chaîne de caractères en variable numérique. Il vous donne également une réponse égale à zéro si la chaîne n'est pas numérique.

Il n'est valable que pour les valeurs numériques positives

entières, mais rien ne vous empêche de l'adapter pour manipuler à la fois les valeurs positives ou négatives, les valeurs entières ou décimales.

Vous remarquerez que le sous-programme affiche zéro si la chaîne contient la valeur "0". Ceci peut être mis à profit dans les cas où la valeur 0 n'est pas désirée.

***LE SOUS-PROGRAMME***

La chaîne de caractères à analyser et à convertir doit être affectée à la variable **A$** pour que le sous-programme puisse fonctionner.

Par ailleurs, la variable **X** contient l'équivalent numérique de la chaîne de caractères si celle-ci est numérique. Sinon X contient la valeur initiale 0.

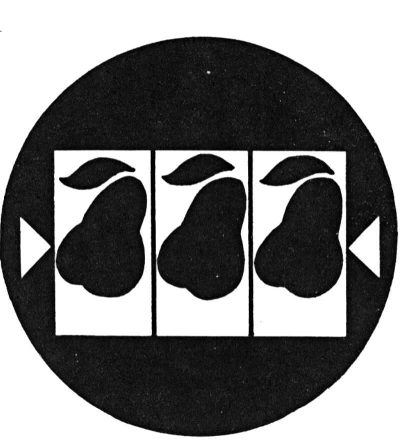
```
9500 REM SET X=VAL(A$)
9510 LET X=0
9520 FOR Y=1 TO LEN A$
9530 IF A$(Y)<"0" OR A$(Y)>"9" THEN GOTO 9570
9540 LET X=X*10+VAL A$(Y)
9550 NEXT Y
9560 RETURN
9570 LET X=0
9580 RETURN
```

L'exemple présenté plus haut a été modifié de façon à faire appel à ce sous-programme. Essayez de le contrecarrer et vous verrez qu'il est très fiable. Vous devrez chercher longtemps avant de le mettre en échéc.

```
100 PRINT "QUEL AGE AVEZ VOUS?"
110 INPUT A$
120 PRINT A$
130 GOSUB 9500
140 IF X<>0 THEN GOTO 200
150 PRINT A$;"? CE N EST PAS UN AGE"
160 GOTO 100
200 ....
....
....
```

## MACHINE A SOUS

***CAPACITE MEMOIRE NECESSAIRE : 1K RAM***



Ce programme vous offre le moyen de tester vos réflexes ; tout dépend de la vitesse à laquelle vous êtes capable de repérer une combinaison gagnante.

Le programme affiche progressivement une série de lignes comportant trois caractères chacune. Le jeu consiste à réunir le maximum de points en appuyant sur une touche quelconque à chaque fois qu'une combinaison gagnante est affichée.

Votre score dépend de la rapidité de vos réflexes et des combinaisons arrêtées.

Les règles sont les suivantes :

* ? ?   donne 2 points
Deux caractères identiques donnent 5 points sauf :

* * ?   qui donne 10 points
Trois caractères identiques donnent 50 points.

Vous pouvez choisir au départ la vitesse de sélection, c'est-à-dire le temps pris par le programme pour déterminer si une touche a été pressée. Une vitesse de 50 vous laisse environ une seconde pour arrêter la combinaison gagnante.

Une fois que vous aurez attrapé le coup, vous choisirez probablement une vitesse inférieure. La valeur 0 (ou même une valeur négative ! ) ne vous permet pas d'arrêter le programme (sauf si vous utilisez la touche BREAK) ; en effet, la fonction **INKEY$** n'aurait pas le temps d'être exécutée.

Appuyez sur la touche "Q" si vous désirez arrêter le jeu et connaître votre total.

Un dernier point qui a son importance : si vous arrêtez une combinaison "non-gagnante", vous <u>perdez</u> dix points !

```
  10 LET S=0                            (initialisation du score
  20 PRINT "VITESSE?"
  30 INPUT F
 100 LET Z$=""                          (boucle principale
 110 FOR X=1 TO 3                       (la roue tourne
 120 LET Z$=Z$+"*£+-=.$"(INT (RND*7)+1)+" "(...et choisit 3 carac-
                                                                tères
 130 NEXT X
 140 SCROLL
 150 PRINT Z$;                          (affiche les caractères
 160 FOR X=1 TO F                       (temps de réponse autorisé
 170 IF INKEY$<>"" THEN GOTO 200
 180 NEXT X
 190 GOTO 100                           (pas de réponse-relance la roue
 200 IF INKEY$="Q" THEN GOTO 1000       (fin du jeu ?
 210 LET W=-10                          (pénalisation
 220 IF Z$(1)="*" THEN LET W=2          (calcule le score
 230 IF Z$(1)=Z$(3) OR Z$(3)=Z$(5) OR Z$(1)=Z$(5)
 240 IF Z$(1)=Z$(3) AND Z$(1)="*" THEN LET W=10
 250 IF Z$(1)=Z$(3) AND Z$(1)=Z$(5) THEN LET W=50
 300 PRINT "DONNE ";W                   (affiche les points
 310 SCROLL
 320 LET S=S+W                          (actualise le total
 330 GOTO 100
1000 PRINT "VOUS MARQUEZ ";S            (fin du jeu
```

Si vous possédez un ZX80, effacez la ligne 180 et modifiez la ligne 160 de la façon suivante :

```
160 PAUSE F
```

Une extension 16K permet d'apporter quelques améliorations à la version de base. En ajoutant les lignes ci-dessous, vous pouvez entre autre déterminer la durée du jeu dès le début.

Le déroulement du jeu est accéléré quand vous marquez des points ; il est au contraire ralenti si vous en perdez. Le jeu consiste à réunir le maximum de points en un certain laps de temps (par exemple : 2 mn).

Les lignes suivantes doivent être ajoutées au programme précédent, sauf la ligne 20 qui vient remplacer la ligne originale :

```
   1 REM 16K MACHINE A SOUS
   2 REM ******************

  20 PRINT "VITESSE INITIALE?"          (change le message

  35 IF F<1 THEN GOTO 20                (vitesse correcte ?

  40 PRINT "DUREE DU JEU? (MN)"         (durée ?
  50 INPUT N
  60 IF N<1 OR N>10 THEN GOTO 40        (durée correcte ?
  70 POKE 16437,128+N*12                (top chrono

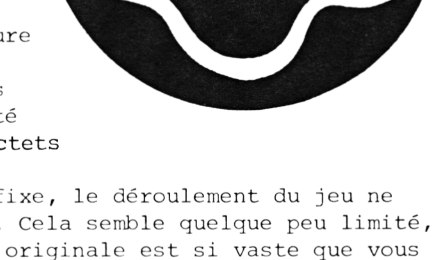
 105 IF PEEK 16437=128 THEN GOTO 2000   (le temps est-il écoulé ?

 325 LET F=F-(W/10)                     (modifie la vitesse du jeu
 327 IF F<1 THEN LET F=1                (vérifie la vitesse

1010 STOP                               (fin du jeu
2000 SCROLL
2010 PRINT "*** TEMPS ECOULE ***"
2020 SCROLL
2030 GOTO 1000
```

# LE ZX 81 ET LE JEU DE L'AVENTURE ALZAN, CITÉ INTERDITE

***CAPACITE MEMOIRE NECESSAIRE : 16K RAM***



Le programme présenté ici vous permet de créer votre propre jeu de l'aventure. L'exemple fourni a pour but de montrer le processus utilisé tout en vous divertissant par la même occasion.

Si vous connaissez déjà toutes les règles du jeu de l'aventure, les paragraphes suivants n'ont pas beaucoup d'intérêt pour vous.

A l'origine, le jeu de l'aventure a été conçu pour des ordinateurs bien plus importants et bien plus rapides que le ZX81, leur capacité mémoire pouvant atteindre 256K octets (de quoi laisser rêveur...).
L'arrangement interne restant fixe, le déroulement du jeu ne varie pas d'une partie à l'autre. Cela semble quelque peu limité, mais de toute façon la version originale est si vaste que vous risqueriez fort de vous en lasser avant de connaître toutes les ficelles.

Comment se présente le jeu lui-même ? Il se déroule dans un labyrinthe où est dispersé un certain nombre de trésors. A vous d'en réunir le plus possible, sachant que certains de ces trésors ne peuvent être pris facilement.
Citons pour exemple la perle enfermée à l'intérieur d'une huître géante ; celle-ci ne s'en sépare pas si facilement et il faut

d'abord trouver l'astuce pour lui faire ouvrir ses deux valves !

Votre emplacement vous est reprécisé à chaque fois que vous jouez ; l'ordinateur attend ensuite que vous entriez deux mots-clés (au maximum).

Par exemple, pour se diriger dans une direction donnée, tapez ALLER SUD, MARCHER EST ou GRIMPER ICI, etc... Pour prendre un objet à portée de la main, tapez par exemple PRENDRE OR, PRENDRE VASE, etc...

Ces objets ne sont pas obligatoirement des trésors mais peuvent s'avérer utiles un peu plus tard au cours du jeu pour franchir un obstacle.

Pour corser le tout, des nains apparaissent aléatoirement çà et là pour vous lancer des poignards. Si l'un de ces poignards vous atteint, vous êtes tué sur le coup ! Vous avez quand même la possibilité de vous défendre avec une hache ou tout autre arme qui vous tombe sous la main.

La version originale étant trop vaste pour être exécutée par le ZX81, nous avons réuni quelques idées d'inspirations différentes pour créer une version adaptable à plusieurs types de jeux.

Vous pouvez ainsi créer votre propre aventure et - pourquoi pas ? - l'échanger avec celle d'un ami.

Le jeu se présente en trois parties :

| | |
|---|---|
| **Le programme principal** | qui contrôle le déroulement du jeu et vérifie si toutes les instructions sont suivies correctement. |
| **Le sous-programme de chargement** | qui vous permet de créer votre propre jeu. |
| **La partie informations** | qui détermine l'originalité de chaque jeu. Elle renferme les descriptions, les mots-clés, les objets, etc... |

## *CREATION DE VOTRE AVENTURE PAS A PAS*

Si vous expérimentez ce jeu pour la première fois, il est préférable de commencer par entrer le programme présenté dans ce chapitre et de l'exécuter. Ceci vous donnera une meilleure idée des règles du jeu et vous permettra de suivre plus facilement les explications qui suivent.

Les différentes étapes à suivre dans la création du jeu sont les suivantes :

1- Entrez en mémoire le programme principal avec le sous-programme de chargement ; ces deux parties constituent la base même de tous les jeux que vous pourrez créer et sont donc à sauvegarder

sur cassette.

2- Entrez en mémoire les messages et les descriptions (vous pouvez utiliser "Alzan, cité interdite" ou le petit exemple-test présenté plus loin).

3- Tapez RUN 9000 pour démarrer le sous-programme de chargement puis entrez en mémoire tous les tableaux et variables correspondant au jeu choisi. Le sous-programme de chargement sauvegarde automatiquement le programme à votre place.

4- Démarrez le jeu en introduisant la commande GOTO 1 : en général cette opération est inutile dans la mesure où le sous-programme de chargement démarre lui-même le jeu une fois le programme principal chargé.

***FONCTIONNEMENT DU PROGRAMME PRINCIPAL***

Les paragraphes suivants semblent assez denses au premier abord ; cependant, vous comprendrez rapidement la conception du programme si vous suivez l'exemple-test pas à pas en vous aidant des commentaires.

Pour que le jeu soit complet, la partie **Informations** (une des trois parties mentionnées un peu plus haut) doit contenir plusieurs rubriques.

Pour vous aider à comprendre la structure de cette partie Informations, n'hésitez pas à vous référer à l'exemple-test présenté plus loin.

Les rubriques nécessaires sont :

## 1- Les descriptions du labyrinthe

Chaque salle du labyrinthe porte un numéro compris entre 1 et le nombre total de salles. Le programme principal effectue un branchement au sous-programme démarrant à la ligne 8000+(salle *10), ce qui affiche la description de la salle correspondante. Ainsi la salle n° 1 est décrite à la ligne 8010, la salle n° 2 à la ligne 8020, la salle n° 3 à la ligne 8030 etc...
Chaque description de salle doit être suivie d'une instruction RETURN (lignes 8015, 8025, 8035 ...).

## 2- Les messages

Certains messages sont directement affichés par le programme principal si besoin est (voir un peu plus loin). Ce type de message peut avoir des motivations très variées ; par exemple, si les mots-clés "ALLUME LAMPE" sont entrés alors que la lampe est déjà allumée, un message approprié peut en avertir le joueur. De même que pour les salles, chaque message porte un

numéro différent : la ligne 7010 contient le message n° 1, la ligne 7020 le message n° 2 etc... Dans ce cas également, chaque message doit être suivi d'une instruction RETURN (7015, 7025...).

## 3- Les objets

Au cours du jeu le joueur a la possibilité d'utiliser un certain nombre d'objets. Ceux-ci peuvent être simplement transportés ou manipulés d'une certaine façon. Chaque objet est caractérisé par deux variables indicées : la première,O(), indique au programme principal l'emplacement initial de l'objet (numéro de salle). La deuxième O$, contient la description de l'objet, de 16 caractères au plus. Cependant, rien ne vous empêche de modifier la longueur maximum de O$ pour un autre jeu si besoin est. Le nombre total d'objets est affecté à la variable O. Un objet qui n'existe pas initialement doit porter un numéro de salle égal à zéro.

## 4- Le vocabulaire

Le tableau de vocabulaire contient une liste de mots-clés reconnaissables par le programme en tant que commandes. Chacun d'eux est accompagné du nombre à deux chiffres qui représente le code dans lequel il sera traduit au moment voulu. Ainsi, deux mots-clés peuvent être traités en même temps. Prenons par exemple les mots "QUITTER" et "SORTIR" : ceux-ci ont la même signification et auront donc le même code à deux chiffres.
Les douze premiers mots-clés sont réservés aux commandes de direction (NORD, SUD, HAUT, BAS...) ; bien entendu, il ne s'agit pas là d'une règle stricte.
Les mots-clés, d'une longueur maximum de six caractères, sont regroupés dans le tableau défini par la variable V$(). Les deux premiers caractères correspondent au "code" à deux chiffres du mot-clé, ce dernier étant représenté par les quatre autres caractères.
Ne pas oublier de faire précéder les codes inférieurs à 10 par un zéro (exemple : 07).
Le nombre total de mots-clés réunis dans le tableau défini par la variable V$() est affecté à la variable V.

## 5- Les couloirs

Les couloirs souterrains faisant communiquer les différentes salles sont regroupés dans le tableau défini par la variable M$(). A chaque salle du labyrinthe correspond une ligne du tableau indiquant le nombre (et la direction) des couloirs permettant de sortir de la salle.
Chaque couloir est défini par quatre caractères ; les deux premiers représentent le code du mot-clé correspondant à la direction. Les deux autres représentent le numéro de la salle où

aboutit ce couloir (ne pas oublier de faire précéder le numéro de salle par un zéro s'il est inférieur à dix). Le code "00" clôt la liste de couloirs pour la salle considérée. Les couloirs de la salle n° 1 sont définis dans le contenu de la variable M$(1). Un exemple pourrait être :

M$(1)="0127061300"

Le code 01 entraînera le programme principal à la salle n° 27 alors que le code 06 l'entraînera à la salle n° 13. Les deux zéros en fin de chaîne closent la liste de couloirs partant de la salle n° 1. La variable R contient le nombre total de salles. La variable M$ contient 32 caractères au maximum, ce qui convient parfaitement dans la majorité des cas.

## 6- Les actions

Regroupées dans le tableau défini par la variable A$, elles ne sont effectuées que lors de l'entrée de certains mots-clés (ou combinaisons de mots-clés). Remarquez que la direction des mouvements est déterminée par "le tableau des couloirs" (voir ci-dessus). Chaque ligne du tableau répond au schéma général suivant :

Code du mot-clé n° 1 (ou 00 si tous les mots conviennent)
Code du mot-clé n° 2 (ou 00 si tous les mots conviennent)
Conditions supplémentaires
Actions à réaliser si toutes les conditions sont réunies.

Un exemple illustrant la dernière partie du tableau pourrait être : "Si les mots-clés "ALLUME LAMPE" sont entrés alors que la lampe est déjà allumée, afficher le message n° 5".
Vous trouverez un peu plus loin toute une liste de conditions et d'actions possibles. D'une façon générale, celles-ci sont de la forme :

AABBC01C02.A01A02A03.

AA et BB représentent le code respectif des deux mots-clés, C01 et C02 les conditions supplémentaires (leur nombre pouvant être supérieur à 2). La deuxième partie de la chaîne de caractères représente les codes correspondant aux actions appropriées (A01, A02, A03 ; rien ne vous empêche d'en ajouter d'autres si vous le désirez). Remarquez que les deux zones sont chacune terminées par un point. Un exemple d'application complet vous est présenté un peu plus loin.
Le nombre total de lignes du tableau est affecté à la variable A.
Chaque ligne du tableau défini par la variable A$() est limitée à 31 caractères au maximum. Si cela se révélait insuffisant, vous pouvez contourner la difficulté en introduisant une action qui "active" un pointeur momentané. D'autres lignes peuvent être

ajoutées au tableau défini par la variable C$() pour contrôler le pointeur qui déclenchera alors l'exécution. L'opération terminée, le pointeur est désactivé. Cette astuce a été mise à profit dans le programme "Alzan, cité interdite".

## 7- Les conditions

Ce tableau, défini par la variable C$() ressemble au précédent ; la seule différence réside dans l'absence de mots-clés. Il est passé en revue à chaque fois qu'une commande est entrée, ce qui vous permet de parer à certaines éventualités (par exemple, l'intervention d'un nain ou l'affichage de certains messages résultant d'une situation particulière). Le nombre total de lignes du tableau est affecté à la variable C. Chaque ligne du tableau est de la forme :

C01C02.A01A02.

C01, C02, A01... sont définis de la même façon que pour le tableau d'actions. Remarquez que des points terminent les deux zones ici également.
Les différentes conditions que vous pourrez utiliser sont représentées par un code à trois caractères ; le premier est une lettre correspondant à la condition voulue. Les deux autres (notésschématiquement nn dans ce qui suit) représentent un paramètre à deux chiffres associé à cette condition.

| Code conditions | Test effectué |
|---|---|
| A | nn est le numéro de la salle en cours |
| B | L'objet nn est présent (ou transporté) |
| C | L'objet nn est absent (ou non transporté) |
| D | L'objet nn est transporté |
| E | Le pointeur nn est "activé" |
| F | Le pointeur nn n'est pas activé |
| G | Le compteur régressif nn a atteint la valeur 1 |
| H | Le nombre aléatoire (compris entre 1 et 99) est inférieur à nn. |

| Code actions | Action effectuée |
| --- | --- |
| A | Affiche la liste des objets transportés |
| B | Transporte l'objet nn |
| C | Pose l'objet nn |
| D | Affiche le texte du message nn (résultat de l'exécution de l'instruction GOSUB "ligne 7000+(nn*10)" |
| E | "Active" le pointeur nn |
| F | Désactive le pointeur nn |
| G | Fixe le compteur régressif nn à la valeur mm |
| H | Inverse les lignes nn et nn+1 dans le tableau des objets |
| I | Pose l'objet nn dans le n° de salle en cours |
| J | Met le n° de salle contenant l'objet nn à "00" |
| K | Fixe le numéro de salle en cours à la valeur nn (c'est-à-dire oblige le jeu à continuer à la salle nn) |
| L | Affiche "D ACCORD" et attend une nouvelle commande |
| M | Attend une nouvelle commande |
| N | Attend une nouvelle commande, mais le tableau des conditions n'est pas passé en revue d'abord |
| O | Affiche la description de la salle en cours |
| P | Interrompt le jeu (si le joueur répond "OUI" à la question "ETES VOUS SUR?", le jeu est arrêté) |
| Q | Interruption du jeu |

Un exemple d'application vous est présenté dans les lignes suivantes. Le mot-clé "PRENDS" a pour code le nombre 15 et le mot-clé "OR" le nombre 22. L'objet "LINGOT" a le code numéro 3. Dans le tableau des actions pourrait figurer une ligne du type :

1522B03.B03L.

Cette ligne signifie que les mots-clés 15 et 22 doivent être entrés (PREND OR) et que la condition B03 doit être satisfaite (c'est-à-dire si l'objet est présent dans la salle ou est transporté). Si tout est en ordre, les actions B03 et L sont exécutées ; l'objet n° 3 est alors transporté. L'ordinateur affiche le message "D ACCORD" et attend une nouvelle commande.

Le programme permet l'utilisation de dix pointeurs différents. Ils peuvent être "activés", "désactivés" ou contrôlés (se reporter au paragraphe Objets). Au début du jeu, ils sont tous à l'état désactivé.

Les pointeurs 1, 2 et 3 ont un rôle particulier dans le programme principal alors que les sept autres restent disponibles. Les trois premiers pointeurs sont :

1 : indique le nombre total d'objets transportés
2 : indique au programme principal si la salle est éclairée ou non, c'est-à-dire si une lampe est nécessaire
3 : indique au programme principal si la lampe est allumée ou non.

Si la salle est plongée dans l'obscurité et que la lampe est éteinte, le programme principal affiche le message suivant :

"ON N Y VOIT RIEN - MIEUX VAUDRAIT ALLUMER POUR EVITER LES ENNUIS"

Le pointeur n° 2 est désactivé quand le joueur est au-dessus du sol, activé quand il est en-dessous.

Il existe également cinq compteurs régressifs à usage général. Le code actions G permet de les initialiser à n'importe quelle valeur à deux chiffres. Deux paramètres doivent être définis avant l'introduction du code G, à savoir le numéro du compteur et sa valeur initiale. Par exemple la ligne G0105 initialise le compteur n° 1 à la valeur 5. Les compteurs 1 à 4 sont automatiquement décrémentés dans certaines conditions définies par le programme principal :

| | |
|---|---|
| Compteur n° 1 | décrémenté à chaque fois qu'une commande est entrée |
| Compteur n° 2 | décrémenté à chaque fois que le pointeur n° 2 est activé (c'est-à-dire quand le jeu se situe dans une salle obscure) |
| Compteur n° 3 | décrémenté à chaque fois que le pointeur est activé alors que le pointeur n° 3 ne l'est pas. Ceci se produit quand le joueur se trouve dans une salle obscure sans lampe allumée. Vous avez ainsi la possibilité de précipiter le joueur dans les oubliettes s'il bouge plus de trois fois dans l'obscurité totale. |
| Compteur n° 4 | décrémenté à chaque fois qu'une commande est entrée (comme pour le compteur n° 1). |

La condition n° 7 contrôle le contenu des compteurs régressifs : dès que l'un d'eux atteint la valeur 1, vous pouvez déclencher une série d'actions.

## Ne pas oublier les deux points suivants :

1- Cinq objets au maximum sont transportables en une seule fois (ligne 4100 du programme principal). Des objets supplémentaires ne peuvent être pris que si, et seulement si, d'autres objets sont posés d'abord.

2- Le programme principal vérifie avant d'exécuter l'action B si le joueur ne transporte pas déjà un objet. De même avant d'exécuter l'action C, il vérifie si le joueur transporte bien un objet. Cela permet d'économiser un nombre de lignes intéressant dans le tableau des actions.

A la fin du listing de ce programme, vous trouverez un exemple-test (portant sur 6 salles seulement). Il vous permet de suivre le déroulement des opérations et de vérifier si votre nouveau programme fonctionne.

```
  1 REM AVENTURE SUR ZX81
  2 REM *****************
 10 DIM S(10)                         (dimensionne la variable pointeur
 20 DIM C(5)                          (dimensionne le compteur régressif
 30 LET SALLE=1                       (initialisation du nombre de salles
 40 DIM P$(2,2)                       (mots-clés 1 et 2
 50 DIM O(O)                          (tableau des objets
 60 FOR X=1 TO O                      (disposition initiale des objets
 70 LET O(X)=Q(X)
 80 NEXT X
 90 RAND
100 IF NOT S(2) THEN GOTO 200         (fait-il noir ?
110 IF C(2) THEN LET C(2)=C(2)-1      (compteur régressif d'obscurité
120 IF S(3) THEN GOTO 200             (lampe allumée ?
130 PRINT "ON N Y VOIT RIEN-MIEUX VAUDRAIT",
          "ALLUMER POUR EVITER LES ENNUIS."
140 IF C(3) THEN LET C(3)=C(3)-1      (comptage régressif si lampe non allumée
150 GOTO 1000                         (attend une commande
200 REM DESCRIPTION DE LA SALLE
210 PRINT
220 GOSUB 8000+SALLE                  (affiche la description de la salle
300 LET F=0                           (réactive le drapeau
310 FOR X=1 TO O                      (affiche les objets présents
320 IF O(X)<>SALLE THEN GOTO 500
330 IF F THEN GOTO 400
340 PRINT ,,"IL Y A AUSSI:"
350 LET F=1
400 PRINT "   ";O$(X)
500 NEXT X
```

```
1000 REM ACCEPTATION DE LA COMMANDE(attend une commande
1010 LET T=1                       (vérifications automatiques prio-
                                                          ritaires
1020 GOTO 2000                     (comptage régressif à chaque com-
                                                             mande
1100 IF C(1) THEN LET C(1)=C(1)-1  (comptage régressif à chaque com-
                                                             mande
1110 IF C(4) THEN LET C(4)=C(4)-1  (comptage régressif
1120 PRINT ,,">"                   (signe d'invite-utiliser le carac-
1130 INPUT Y$                      (entrée de la commandetère inverse
1140 CLS
1150 LET Y=0                       (analyse la commande
1160 PRINT ">";Y$                  (affiche la commande en haut
1170 LET P$(2)="00"
1200 FOR W=1 TO 2                  (contrôle jusqu'à 2 mots-clés
1210 GOSUB 6000
1220 IF Y>=LEN Y$ THEN GOTO 1300   (tout est analysé ?
1230 IF P$(W)="00" THEN GOTO 1210  (le mot-clé est-il trouvé ?
1240 NEXT W                        (mot-clé suivant
1300 IF P$(1)<>"00" THEN GOTO 1600(y a-t-il au moins un mot ?
1310 PRINT " PARDON?"              (affiché si rien n'est trouvé
1320 GOTO 100                      (essaye encore
1600 REM CONTROLE DU MOUVEMENT
1610 LET Z=1                       (analyse à présent le tableau des
                                                        mouvements
1620 LET T$=M$(SALLE)(Z TO Z+1)    (donne mot-clé correspondant
1630 IF T$="00" THEN GOTO 1900     (fin de ligne ?
1640 IF T$<>P$(1) THEN GOTO 1700   (vérifie la correspondance avec le
1650 LET SALLE=VAL (M$(SALLE)(Z+2 TO Z+3))               mot-clé n° 1
1660 GOTO 100                      (continue dans une nouvelle salle
1700 LET Z=Z+4                     (essaye la correspondance suivante
1710 GOTO 1620
1900 LET T=0                       (active le drapeau "Actions"
1910 LET CORRESP=0                 (pas de correspondance encore
                                                           trouvée
2000 REM CONTROLE DES CONDITIONS
2010 LET CP=0                      (initialise numéro de tableau
2100 LET CP=CP+1
2110 IF NOT T THEN GOTO 2300       (analyse le tableau des actions ?
2120 LET E$=C$(CP)                 (tableau des conditions
2130 GOTO 2600
2300 IF CP<=A THEN GOTO 2400       (tout a été analysé ?
2310 IF CORRESP THEN GOTO 1000     (correspondance trouvée ?
2320 PRINT "IMPOSSIBLE";           (affiche le message
2330 IF VAL (P$(1))<13 THEN PRINT "PRENDRE CETTE DIRECTION";
2340 PRINT "."
2350 GOTO 100                      (essaye encore
2400 IF A$(CP)(1 TO 2)<>P$(1) THEN GOTO 2100
                                   (vérifie la correspondance avec le
                                                       mot clé n° 1
```

```
2410 LET Y$=A$(CP)(3 TO 4)                       (mot-clé n° 2
2420 IF Y$<>"00" AND Y$<>P$(2) THEN GOTO 2100
2430 LET E$=A$(CP)(5 TO )                        (obtient conditions ou actions
2600 REM CONDITIONS
2610 LET E=1                                     (analyse à présent les conditions
2700 IF E$(E)="." THEN GOTO 3000                 (un point clot les conditions
2710 LET TYPE=CODE (E$(E))-38                    (obtient le code de la condition
2720 LET N=VAL (E$(E+1 TO E+2))                  (obtient le paramètre
2800 GOSUB 2900+TYPE*10                          (détermine si vrai ou faux
2810 IF NOT OK THEN GOTO 2100
2820 LET E=E+3                                   (essaye la condition suivante
2830 GOTO 2700
2900 LET OK= (N=SALLE)                           (condition A-se reporter au texte
2905 RETURN
2910 LET OK=(O(N)=SALLE OR O(N)<0)               (condition B
2915 RETURN
2920 LET OK=(O(N)<>SALLE AND O(N)>=0) (condition C
2925 RETURN
2930 LET OK=(O(N)<0)                             (condition D
2935 RETURN
2940 LET OK=S(N)                                 (condition E
2945 RETURN
2950 LET OK=(NOT S(N))                           (condition F
2955 RETURN
2960 LET OK=(C(N)=1)                             (condition G
2965 RETURN
2970 LET OK=((INT (RND*100)+1)<=N)               (condition H
2975 RETURN
3000 REM ACTIONS
3010 LET CORRESP=1                               (exécute à présent les actions
3020 LET E=E+1
3100 IF E$(E)="." THEN GOTO 2100                 (terminé ?
3110 LET TYPE=CODE (E$(E))-38                    (obtient le code action
3120 IF E$(E+1)<>"." THEN LET N=VAL (E$(E+1 TO E+2))
                                                 (obtient un paramètre
3200 LET BREAK=0                                 (retourne au numéro de ligne
3210 GOSUB 4000+TYPE*100                         (exécute l'action
3220 IF BREAK THEN GOTO BREAK                    (va à la ligne considérée
3230 LET E=E+3                                   (action suivante
3240 GOTO 3100
4000 PRINT                                       (action A - voir tableau
4010 PRINT "VOUS TENEZ:"
```

```
4020 LET F=1
4030 FOR X=1 TO O
4040 IF O(X)>=0 THEN GOTO 4070
4050 PRINT "    ";O$(X)
4060 LET F=0
4070 NEXT X
4080 IF F THEN PRINT "   RIEN."
4090 LET BREAK=100
4095 RETURN
4100 IF S(1)<5 THEN GOTO 4140          (action B
4110 PRINT "VOUS NE POUVEZ PORTER PLUS"
4120 LET BREAK=100
4130 RETURN
4140 IF O(N)=-1 THEN GOTO 4180
4150 LET O(N)=-1
4160 LET S(1)=S(1)+1
4170 RETURN
4180 PRINT "VOUS L AVEZ DEJA"
4190 GOTO 4120
4200 IF O(N)=-1 THEN GOTO 4240          (action C
4210 PRINT "VOUS N AVEZ PAS ";O$(N)
4220 LET BREAK=100
4230 RETURN
4240 LET O(N)=SALLE
4250 LET S(1)=S(1)-1
4260 RETURN
4300 PRINT                              (action D
4310 GOSUB 7000+N*10
4320 RETURN
4400 LET S(N)=1                         (action E
4410 RETURN
4500 LET S(N)=0                         (action F
4510 RETURN
4600 LET C(N)=VAL (E$(E+3 TO E+4))      (action G
4610 LET E=E+2
4620 RETURN
4700 LET X=O(N)                         (action H
4710 LET O(N)=O(N+1)
4720 LET O(N+1)=X
4730 RETURN
4800 LET O(N)=SALLE                     (action I
4810 RETURN
4900 IF O(N)<0 THEN LET S(1)=S(1)-1     (action J
```

```
4910 LET O(N)=0
4920 RETURN
5000 LET SALLE=N                              (action K
5010 RETURN
5100 PRINT "D ACCORD"                         (action L
5200 LET BREAK=1000                           (action M
5210 RETURN
5300 LET BREAK=1100                           (action N
5310 RETURN
5400 LET BREAK=100                            (action O
5410 RETURN
5500 PRINT "ETES VOUS SUR? ";                 (action P
5510 INPUT W$
5520 PRINT W$
5530 IF CHR$ CODE W$<>"Y" THEN RETURN
5600 GOTO 9999                                (action Q
6000 REM ANALYSE LE MOT
6010 DIM W$(4)                                (quatre premières lettres
6015 LET P$(W)="00"                           ("réponse" obtenue si pas de
                                                         réponse trouvée
6020 GOSUB 6600                               (trouve le premier caractère
6025 IF FIN THEN RETURN                       (fin de commande ?
6030 FOR Q=1 TO 4                             (obtient quatre lettres
6040 LET W$(Q)=Y$(Y)
6050 GOSUB 6500                               (vérifie si c'est la fin du
                                                                     mot
6060 IF FIN THEN GOTO 6100
6070 NEXT Q
6080 GOSUB 6500                               (cherche la fin du mot
6090 IF NOT FIN THEN GOTO 6080
6100 IF W$="    " THEN RETURN                 (aucun mot entré
6110 FOR Q=1 TO V                             (analyse le tableau de vocabu-
6120 IF W$=V$(Q)(3 TO ) THEN GOTO 6200                               laire
6130 NEXT Q
6140 RETURN                                   (inconnu dans ce tableau
6200 LET P$(W)=V$(Q)( TO 2)                   (obtient le code du mot-clé
6210 RETURN
6500 LET Y=Y+1                                (vérifie si fin de mot
6510 LET FIN=(Y>LEN Y$)
6520 IF FIN THEN RETURN
6530 LET FIN=(Y$(Y)=" ")
6540 RETURN
6600 LET Y=Y+1                                (cherche la fin du mot
6610 LET FIN=(Y>LEN Y$)
6620 IF FIN THEN RETURN
6630 IF Y$(Y)=" " THEN GOTO 6600
```

```
6640 RETURN
7000 REM MESSAGES ACTIONS
7001 REM MESSAGE NO.1 OBTENU PAR
7002 REM GOSUB LIGNE  7010
7999 RETURN
8000 REM DESCRIPTION DES SALLES
8001 REM SALLE 1 OBTENUE PAR
8002 REM GOSUB LIGNE  8010
9999 STOP
```

Avant de commencer à jouer, vous devez encore adjoindre deux parties au programme principal, celui-ci constituant en quelque sorte le "cerveau du jeu". La première partie est le "sous-programme de chargement" du jeu ; celui-ci initialise certaines variables et la totalité des tableaux (tableaux des objets, des couloirs, de vocabulaire, d'actions et de conditions).

Le jeu lui-même constitue la deuxième partie ; celle-ci a été étudiée pour être rapidement adaptée à vos versions personnelles.

Deux mini-aventures vous sont proposées un peu plus loin. Leur chargement et leur exécution mettent en évidence les relations existant entre les différentes parties de ce programme de jeu (sans compter le plaisir de jouer !).

## LE SOUS-PROGRAMME DE CHARGEMENT

Placé à la suite du programme principal, ce sous-programme peut être effacé une fois les tableaux entrés en mémoire. Cependant, il est beaucoup plus prudent de ne pas effectuer cette dernière opération avant de s'assurer que tout est correct.

Imaginez votre désarroi si vous avez effacé le sous-programme alors que des mots ont été oubliés dans le tableau de vocabulaire (ou tout autre tableau).

Lors de son exécution le sous-programme vous demande en premier lieu combien de lignes doit comporter chaque tableau (nombre de mots dans le tableau de vocabulaire par exemple). La réponse fournie détermine la dimension des tableaux avant l'introduction des lignes (une par une).

Un arrêt entre chaque tableau créé permet d'effectuer la vérification des éléments entrés. Si le contenu d'un tableau s'avère incorrect, il est alors inutile de tout retaper ; il suffit de remplacer le tableau incriminé.

```
9000 REM CHARGEMENT DES TABLEAUX
9010 CLS
9020 PRINT "NOMBRE D OBJETS?"
9030 INPUT O
9040 DIM Q(O)
9050 DIM O$(O,16)
9080 FOR X=1 TO O
9090 SCROLL
9100 PRINT "SALLE NO. ";X;" ?"
9110 INPUT Q(X)
9120 PRINT Q(X)
9130 SCROLL
9140 PRINT "DESCRIPTION?",
9150 INPUT O$(X)
9160 PRINT O$(X)
9170 NEXT X
9199 STOP
9200 CLS
9210 PRINT "NOMBRE DE MOTS?"
9220 INPUT V
9230 DIM V$(V,6)
9240 FOR X=1 TO V
9250 SCROLL
9260 INPUT V$(X)
9270 PRINT V$(X)
9280 NEXT X
9299 STOP
9300 CLS
9310 PRINT "NOMBRE DE SALLES?"
9320 INPUT R
9330 DIM M$(R,32)
9340 FOR X=1 TO R
9350 SCROLL
9360 INPUT M$(X)
9370 PRINT M$(X)
9380 NEXT X
9399 STOP
9400 CLS
```

```
9410 PRINT "NOMBRE DE CONDITIONS?"
9420 INPUT C
9425 LET C=C+1
9430 DIM C$(C,21)
9440 FOR X=1 TO C-1
9450 SCROLL
9460 INPUT C$(X)
9470 PRINT C$(X)
9480 NEXT X
9490 LET C$(C)=".N."
9499 STOP
9500 CLS
9510 PRINT "NOMBRE D ACTIONS?"
9520 INPUT A
9530 DIM A$(A,31)
9540 FOR X=1 TO A
9550 SCROLL
9560 INPUT A$(X)
9570 PRINT A$(X)
9580 NEXT X
9599 STOP
9600 CLS
9610 PRINT "ENTRER LE JEU DE L AVENTURE"
9620 INPUT N$
9630 PRINT ,,"DEMARRER LA CASSETTE..."
9640 PAUSE 150
9650 CLS
9660 SAVE N$
9670 GOTO 1
9999 STOP
```

Avant de vous lancer au coeur de l'action, entrez l'exemple-test suivant. Il vous donnera une idée du fonctionnement de ce programme et vous montrera comment créer le vôtre éventuellement. Il vous permet également de tester si des erreurs ont été introduites lors du chargement.

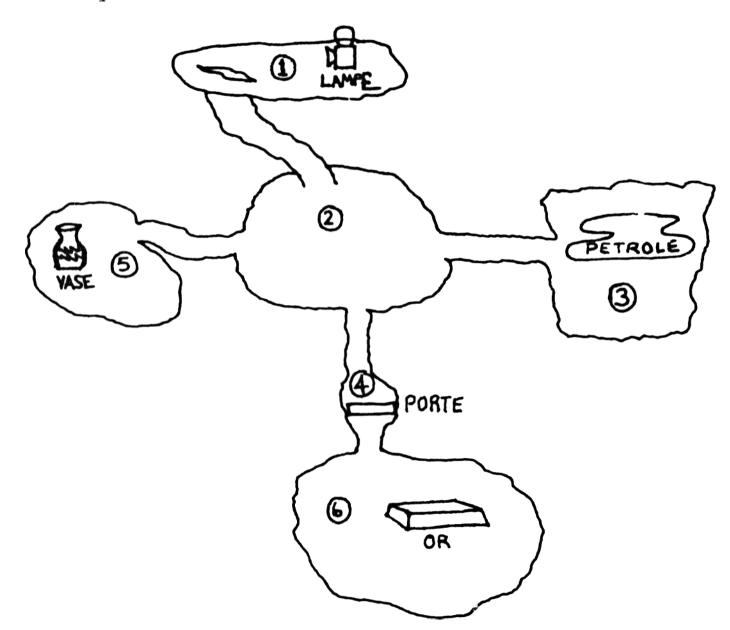
***EXEMPLE-TEST***

Ce petit exemple d'application du jeu de l'aventure prend place dans un labyrinthe comprenant six salles. Une lampe a été placée dans la salle n° 1, seule salle située au-dessus du sol. Le but consiste à récupérer le lingot d'or entreposé dans la cave n° 6 pour le ramener à la salle n° 1.

Malheureusement, pour cela il faut franchir une porte rouillée (cave n° 4) qui ne veut pas s'ouvrir.

La cave n° 5 renferme un vase alors que la cave n° 6 contient une mare d'huile. Manifestement la seule action logique consiste à remplir le vase d'huile pour aller graisser les gonds de la porte ! L'ouvrir et prendre le lingot devient alors un jeu d'enfant.

Plan du labyrinthe :




Les pointeurs 2 et 3 servent à représenter respectivement "l'obscurité des salles" et la position allumée ou éteinte de la lampe. Si vous posez la lampe allumée quelque part, elle est automatiquement pointée comme éteinte ; vous ne pouvez donc la laisser dans un coin pendant que vous errez dans le souterrain.

Le pointeur n° 5 indique si la porte a été huilée ; le pointeur n° 6 est mis à un lorsque la porte est ouverte.

Entrez les messages et la description des salles, puis la commande RUN 9000 pour démarrer le sous-programme de chargement ; celui-ci entre sous forme de tableaux les objets, le vocabulaire, les couloirs, les conditions et les mots-clés correspondant aux actions.

Attention ! Une fois que vous avez tout chargé en mémoire, **SAUVEGARDEZ LE PROGRAMME sur cassette (SAVE)!!!**

Lancez l'exécution du programme en entrant la commande GOTO 1 (sinon le contenu de toutes les variables serait effacé). En principe, si vous suivez le processus indiqué ci-dessus, le programme principal doit être entré sans problèmes sérieux.

Une fois que vous aurez tout bien en mains, rien ne vous empêchera plus d'imaginer vos propres jeux.

Un détail important : le tableau des couloirs ne comporte pas de tunnels reliant les salles 1 et 2. A la place, une ligne contenant le mot-clé approprié (06) a été ajouté dans le tableau des actions. Par ce moyen il est possible de s'assurer que le pointeur "d'obscurité" est mis à 1 chaque fois que le mot-clé "DESCENDRE" est entré au niveau de la salle n° 1.

De la même manière, il n'y a pas lieu d'entrer un couloir reliant les salles 4 et 6 puisque la porte bloque le passage.

Par conséquent, une ligne supplémentaire est entrée dans le tableau des actions (03 00 A04 F06....) ; celle-ci contrôle le pointeur n° 6 chaque fois que le mot "SUD" est entré au niveau de la salle n° 4.

En règle générale, si vous voulez poser certaines conditions au joueur quand il passe d'une salle donnée à une autre, préférez le tableau des actions à celui des couloirs.

***Messages*** :

```
7010 PRINT "LA PORTE EST COINCEE"
7015 RETURN
7020 PRINT "LA PORTE EST OUVERTE"
7025 RETURN
7030 PRINT "C EST DEJA ALLUME"
7035 RETURN
7040 PRINT "VOUS REUSSISSEZ TANT BIEN",
           "QUE MAL A OUVRIR LA PORTE."
7045 RETURN
7050 PRINT "C EST TROP DUR A OUVRIR",
           "POUR VOUS."
7055 RETURN
7060 PRINT "VOUS AVEZ REUSSI. BRAVO."
7065 RETURN
7070 PRINT "VOUS NE POUVEZ FRANCHIR LA PORTE"
```

```
7075 RETURN
```

***Description des salles :***

```
8010 PRINT "VOUS ETES AU BORD D UN GOUFFRE."
8015 RETURN
8020 PRINT "VOUS ETES DANS UN LABYRINTHE",
           "AVEC DES PASSAGES MENANT VERS",
           "EST,SUD,OUEST. UN PASSAGE",
           "OBSCUR GRIMPE DERRIERE VOUS."
8025 RETURN
8030 PRINT "CETTE CAVE CONTIENT SEULEMENT",
           "UNE MARE D HUILE."
8035 RETURN
8040 PRINT "VOICI UNE GRANDE PORTE ROUILLEE"
8045 RETURN
8050 PRINT "VOUS ETES DANS LA CHAMBRE OUEST."
8055 RETURN
8060 PRINT "VOUS ETES DANS LA CAVE AU TRESOR"
8065 RETURN
```

***Objets :***

Nombre d'objets : 5

| N° | Numéro de salle | Description |
|---|---|---|
| 1 | 1 | UNE LAMPE |
| 2 | 0 | UNE LAMPE ALLUMEE |
| 3 | 5 | UN VASE CHINOIS |
| 4 | 0 | UN VASE D HUILE |
| 5 | 6 | UN LINGOT D OR |

***Vocabulaire :***

*Nombre de mots : 25*

Chaque ligne comprend six caractères maximum, les deux premiers correspondant au numéro du mot.

```
01N
01NORD
02E
02EST
03S
03SUD
04O
04OUES
05M
05MONT
06D
06DESC
13PREN
14POSE
15VASE
16OR
17PORT
18OUVR
19LAMP
20ALLU
21REMP
22HUIL
23INVE
24QUIT
25REGA
```

*Nombre de salles : 6*

Tableau des couloirs :

(1) 00
(2) 02030304040500
(3) 040200
(4) 010200
(5) 020200
(6) 010400

*Nombre de conditions supplémentaires : 3*

Conditions (ne pas entrer les espaces !) :

| | |
|---|---|
| A04 E06. D02 N. | (salle n° 4, pointeur n° 6 activé -<br>(signifie que la porte est ouverte |
| A04 F06. D01 N. | (salle n° 4, pointeur n° 6 non activé -<br>(signifie que la porte est fermée |
| A01 D05. D06 Q. | (salle n° 1, objet 5 transporté -<br>(le lingot est sorti - gagné ! |

*Nombre de mots-clés correspondant aux actions : 21.*

***Mots-clés correspondant aux actions :***

| | |
|---|---|
| 13 19 B01. B01 L. | (PRENd la lampe - objet 01 |
| 14 19 B01. C01 L. | (POSE la lampe |
| 13 19 B02. B02 E03 L. | (PRENd la lampe (allumée) - objet 02 -<br>(active aussi le pointeur n° 3 |
| 14 19 B02. C02 F03 L. | (POSE lampe allumée - désactive le<br>(pointeur n° 3 |
| 20 00 D01. H01 E03 L. | (ALLUme la lampe - intervertit les objets 1<br>(et 2 ; active aussi le pointeur n° 3 |
| 20 00 B02. D03 M. | (ALLUme la lampe ; l'objet 02 est déjà là -<br>(affiche 03 |
| 06 00 A01. E02 K02 O. | (DESCend si dans la salle n° 1 - active alors<br>(le pointeur d'obscurité 2 et continue à<br>(la salle n° 2 |
| 05 00 A02. F02 K01 O. | (MONTe si dans salle n° 2 - désactive alors<br>(le pointeur d'obscurité n° 2 et continue à<br>(la salle n° 1 |
| 13 15 B03. B03 L. | (PRENd vase |
| 14 15 B03. C03 L. | (POSE vase |
| 13 16 B05. B05 L. | (PRENd l'or |
| 14 16 B05. C05 L. | (POSE l'or |
| 21 00 B03 A03. H03 L. | (REMPlit le vase - doit avoir lieu dans<br>(la salle n° 3 avec l'objet 3<br>(intervertit les objets 3 et 4 |

```
22 00 A04 B04. H03 E05 L. (HUILe la porte - doit avoir lieu dans la
                          (salle n° 4 avec l'objet 4 (vase plein) -
                          ("vide" la bouteille et active pointeur n° 5
18 00 A04 E05. D04 E06 M. (OUVRe la porte qui doit être huilée -
                          (signifie que le pointeur n° 5 doit être
                          (activé et que l'on est dans la salle n° 4
                          (active le pointeur n° 6
18 00 A04 F05. D05 M.     (essaye d'OUVRir la porte si pas huilée
                          (- affiche le message 5.
03 00 A04 F06. D07 M.     (SUD si pointeur n° 6 non activé -
                          (signifie porte fermée
03 00 A04 E06. K06 O.     (SUD si porte ouverte (pointeur n° 6
                          (activé). Continue à la salle n° 6.
23 00 .A.                 (donne l'INVEntaire
24 00 .P.                 (QUITte le programme
25 00 .O.                 (REGArde où nous sommes
```

***VOS AVENTURES A L'EPREUVE***

Que faire si votre programme ne fonctionne pas ? Les quelques conseils qui suivent ont pour but de vous aider à dépister les erreurs éventuelles.

L'expérience prouve que la plupart du temps les anomalies proviennent du tableau de conditions ou d'actions (spécification des actions incorrecte ou absence de renseignements appropriés).

Vous pouvez exécuter le programme en mode lent ou rapide ; cependant, la recherche de l'action correspondant au mot-clé dans le tableau fait parfois perdre de précieuses secondes. Aussi, le mode rapide est-il préférable dans ce cas.

Si au bout d'une ou deux minutes rien ne se passe, l'anomalie se situe probablement au niveau du tableau des conditions.

Appuyez alors sur la touche BREAK et vérifier une par une les variables suivantes en utilisant la commande PRINT en mode direct :

**CP** contient le numéro de ligne en cours du tableau de conditions ou d'actions.

**T** indique si le tableau de conditions ou d'actions est analysé. La valeur 0 correspond au tableau d'actions, la valeur 1 au tableau de conditions.

**E$** contient une "copie" de la ligne du tableau de conditions ou d'actions.

**P$(1)** contient le numéro du premier mot-clé rencontré après l'introduction d'une commande.

| | |
|---|---|
| **PS(2)** | contient le numéro du deuxième mot rencontré ou "00" si un deuxième mot-clé n'a pas été entré. |
| **SALLE** | numéro de la salle en cours |
| **S(n)** | active le pointeur n/ 0:éteint, 1: allumé |
| **C(n)** | compteur régressif n/ 0:compteur non branché |
| **O(n)** | numéro de la salle renfermant l'objet n. Si l'objet est transporté, O(n) prend la valeur -1 ; si l'objet n'existe pas, O(n) prend la valeur 0. |

Le sous-programme de chargement entre les tableaux (ou variables) suivants :

| | |
|---|---|
| **M$** | tableau des couloirs |
| **R** | nombre de salles |
| **O** | nombre d'objets |
| **Q()** | tableau de localisation des objets ; copié dans O() |
| **O$()** | description des objets |
| **V** | nombre de mots |
| **V$()** | tableau de vocabulaire |
| **C** | nombre des conditions |
| **C$** | tableau des conditions |
| **A** | nombre d'actions |
| **A$** | tableau des actions |

Une erreur courante consiste à employer le code action M à la place du code N pour terminer une entrée dans le tableau des conditions. Le code action M redemande l'analyse du tableau des conditions ; les mêmes conditions étant probablement toujours réalisées, le programme cyclera indéfiniment. Vérifiez que toutes les conditions entrées soient bien suivies du code N, à moins d'avoir une bonne raison pour en utiliser un autre (voir tableaux de l'exemple-test).

## ALZAN, CITE INTERDITE

***CAPACITE MEMOIRE NECESSAIRE : 16K RAM***

Passons maintenant au vif du sujet. Le programme présenté ici est basé sur une adaptation du programme principal.

L'action se déroule dans une ville imaginaire baptisée Alzan. Celle-ci est bâtie au sommet d'une falaise abrupte et abrite voleurs et coupeurs de tête.



Malheureusement pour vous, vous êtes en plein milieu de cette charmante cité ; vous devez trouver le moyen de fuir à tout prix avant d'être pris ou d'avoir contracté la peste. Pour vous faciliter la tâche, la cité est entourée de murs infranchissables (ou presque !) ; à vous de trouver comment les escalader.

Peut-être découvrirez-vous le processus utilisé dans la création du jeu (donc le moyen de gagner à coup sûr !) lorsque vous chargerez le programme.

Aussi, pour conserver un certain suspense essayez de taper le programme sans trop l'approfondir au départ.

Lors de l'exécution du programme, il reste 4150 octets vides. Ceci vous donne une idée de la place dont vous disposez pour écrire vos propres jeux de l'aventure. A noter que l'utilisation du sous-programme de chargement des codes machine (voir chapitre "Utilisation des codes machine" page 79) peut s'avérer très intéressante dans la conception de vos programmes.

Entrez les messages suivants :

```
7010 PRINT "MON PAUVRE...VOUS AVEZ DU",
           "ATTRAPER LA PESTE DANS UNE",
           "TOMBE CAR VOUS VENEZ DE MOURIR"
7015 RETURN
7020 PRINT TAB 12;"AIE AIE AIE"
7022 PRINT "EL BANDITO LE TERRIBLE VOLEUR",
           "VIENT DE RAFLER VOTRE ARGENT",
           "AVANT DE S ENFUIR DANS LA BRUME"
```

```
7025 RETURN
7030 PRINT """AU VOLEUR"" CRIE L OUVREUR",
          "MAIS VOUS REUSSISSEZ A FUIR."
7035 RETURN
7040 PRINT "LA TRAPPE EST DEJA OUVERTE"
7045 RETURN
7050 PRINT "CE N EST PAS DANS VOS MOYENS"
7055 RETURN
7060 PRINT "CA IRA TRES BIEN, MONSIEUR."
7065 RETURN
7070 PRINT " LA TRAPPE EST OUVERTE"
7075 RETURN
7080 PRINT " LA TRAPPE EST FERMEE"
7085 RETURN
7090 PRINT "LE MARCHAND EST PLUS FORT",
          "QUE VOUS..."
7095 RETURN
7100 PRINT "IL VOUS FAUT UNE ECHELLE POUR",
          "FRANCHIR LE MUR."
7105 RETURN
7110 PRINT "C EST DEJA ALLUME"
7115 RETURN
7120 PRINT "QUEL TRAIT DE GENIE"
7125 RETURN
7130 PRINT " VOUS AVEZ ROULE LES GARDIENS ET";
          "SUBTILISE UNE LIASSE DE BILLETS.";
          "PERSONNE N A RIEN REMARQUE.",
          "(DROLES DE TYPES LES ALZANAIS)"
7135 RETURN
7140 PRINT "VOUS AVEZ TOUT PRIS"
7145 RETURN
7150 PRINT "JE NE VOIS PAS DE TORCHE?"
7155 RETURN
7160 PRINT "LE CINEMA EST RESERVE A UNE",
          "UTILISATION PRIVEE."
7165 RETURN
8010 PRINT TAB 8;"BIENVENUE A ALZAN",,,
          "VOUS DEVEZ ESCALADER LES MURS SI";
          "VOUS VOULEZ ECHAPPER AUX VOLEURS";
          "ET AUX COUPEURS DE TETES."
8015 RETURN
8020 PRINT "VOUS ETES DANS LA GRANDE RUE",
          "DEVANT UN BAZAR. LA RUE VA",
          "DE L EST A L OUEST ET UNE",
          "RUELLE SE DIRIGE AU NORD A COTE",
          "DE LA BOUTIQUE."
8025 RETURN
```

```
8030 PRINT "VOUS ETES DANS LA BOUTIQUE. LE",
          "MARCHAND A L AIR ROUBLARD MAIS",
          "IL A UN BEL ETALAGE."
8035 RETURN
8040 PRINT "VOUS ETES DANS UNE RUELLE",
          "DERRIERE LES BUILDINGS. IL Y A",
          "BEAUCOUP DE POUBELLES SOUS",
          "L ESCALIER DE SECOURS."
8045 RETURN
8050 PRINT "VOUS ETES DANS L ESCALIER",
          "DE SECOURS QUI MENE DANS",
          "LE BUILDING PAR UNE PORTE."
8055 RETURN
8060 PRINT "VOUS ETES DESCENDU DANS LA",
          "BOUTIQUE PAR UN ESCALIER DEROBE"
8065 RETURN
8070 PRINT "VOUS ETES SUR UNE COURSIVE",
          "ENTRE LES BUILDINGS."
8075 RETURN
8080 PRINT "CE MUR FAIT PARTIE DE L ENCEINTE";
          "IL Y A UNE PORTE ABANDONNEE",
          "A CET ENDROIT."
8085 RETURN
8090 PRINT "VOUS ETES A UN CARREFOUR."
8095 RETURN
8100 PRINT "VOICI LE MUR D ENCEINTE. LA MER",
          "EST DANS LA BRUME,CE QUI REND",
          "LA VISIBILITE MAUVAISE."
8105 RETURN
8110 PRINT "VOUS VOUS ECRASEZ SUR UN ROCHER",
          "APRES UN PLONGEON DE 100 M DU",
          "HAUT DES REMPARTS. VOUS AUREZ",
          "PLUS DE CHANCE LA PROCHAINE FOIS"
8115 RETURN
8120 PRINT "VOUS ETES DEVANT LA BANQUE"
8125 RETURN
8130 PRINT "LES NOMBREUX GARDIENS DE LA",
          "BANQUE ONT L AIR DE S ENNUYER"
8135 RETURN
8140 PRINT "VOUS ETES DANS UN CUL DE SAC",
          "MAIS IL Y A UNE TRAPPE SUR",
          "LA ROUTE..."
8145 RETURN
8150 PRINT "VOUS ETES DANS UNE CAVITE",
          "SOUS LA TRAPPE. UN PASSAGE",
          "MENE VERS LE SUD."
8155 RETURN
```

```
8160 PRINT "LE PASSAGE MENE DANS UNE",
         "ANCIENNE TOMBE OU SONT",
         "ENTREPOSES DES SARCOPHAGES."
8165 RETURN
8170 PRINT "L OUVREUR NE VOUS LAISSE PAS",
         "ENTRER ALORS QUE LE PROGRAMME",
         "EST COMMENCE. IL VOUS BARRE LE",
         "PASSAGE AVEC SA TORCHE."
8175 RETURN
8180 PRINT "VOUS ETES DEVANT LE CINEMA.",
         "ON ENTEND DES COUPS DE FEU",
         "A L INTERIEUR."
8185 RETURN
8190 PRINT TAB 8;"*** FELICITATIONS ***",
         "VOUS ETES SORTI DE LA VILLE.",
         "C EST EXCEPTIONNEL. BRAVO."
8195 RETURN
```

Tapez à présent la commande RUN 9000 pour démarrer le sous-programme d'initialisation. Le contenu des tableaux est le suivant :

***Nombre d'objets : 11***

| Salle contenant l'objet | Description |
|---|---|
| 0 | UNE LAMPE ALLUMEE |
| 0 | UNE TORCHE |
| 3 | UNE ECHELLE |
| 3 | UN MARTEAU |
| 0 | UN MARTEAU |
| 0 | UNE LIASSE DE BILLETS |
| 0 | UNE TRAPPE |
| 15 | UN SAC DE CLOUS |
| 16 | UNE CARTE DE CREDIT |
| 0 | UNE ECHELLE RAIDE |
| 4 | DU BOIS |

***Nombre de mots : 41***

```
01N          19 MART
01NORD       20LIAS
02E          20BILL
02EST        22SAC
03S          22CLOU
03SUD        23CART
04O          05ESCA
04OUES       05GRIM
05M          29OUVR
05MONT       29SOUL
06D          30FABR
06DESC       30CONS
```

13PREN
14METT
14POSE
15ENTR
15DEDA
16DEHO
16SORT
17TORC
18ECHE
31ALLU
32ACHE
33BOIS
34DERO
34VOLE
35INVE
36QUIT
37REGA

**Nombre de salles : 19**

| Numéro de salle | Couloirs |
|---|---|
| 1 | 00 |
| 2 | 01 04 02 09 04 18 00 |
| 3 | 00 |
| 4 | 02 02 05 05 00 |
| 5 | 06 04 04 07 00 |
| 6 | 00 |
| 7 | 01 08 03 05 00 |
| 8 | 03 07 00 |
| 9 | 01 12 02 10 03 14 04 02 00 |
| 10 | 04 09 00 |
| 11 | 00 |
| 12 | 02 09 04 18 00 |
| 13 | 00 |
| 14 | 01 09 00 |
| 15 | 03 16 00 |
| 16 | 01 15 00 |
| 17 | 00 |
| 18 | 01 12 02 02 00 |
| 19 | 00 |

**Nombre de conditions : 9**

Tableau des conditions (les espaces servent simplement à obtenir une écriture aérée et ne doivent pas êtres entrés) :

```
A01. K02 O.
A16 H30. G0121.
G01. D01 Q.
B06 H10. D02 J06.
A14 E07. D07 N.
A14 F07. D08 N.
A11. Q.
A19. Q.
A06. K03 O.
```

**Nombre d'actions : 47**

Tableau des actions (les espaces servent simplement à obtenir un affichage plus aéré et ne doivent pas être entrés) :

```
13 17 B01. B01 E03 L.
13 17 A17 C01 C02. I02 B02 D03 K18 E10 O.
32 18 B03. D05 N.
13 19 B05. B05 L.
13 20 B06. B06 L.
29 00 A14 E07. D04 N.
29 00 A14. E07 M.
13 22 B08. B08 L.
13 23 B09. B09 L.
14 17 B01. C01 F03 L.
14 17 B02. C02 L.
14 19 B05. C05 L.
14 20 B06. C06 L.
14 22 B08. C08 L.
14 23 B09. C09 L.
05 00 A10 C10. D10 M.
05 00 A08 C10. D10 M.
05 00 A10. K11 O.
05 00 A08. K19 O.
05 00 A15. F02 K14 O.
06 00 A14. E02 K15 O.
31 00 D02. H01 E03 L.
31 00 B01. D11 N.
32 19 B04 B06. H04 J06 B05 L.
32 19 B04 B09. H04 D06 B05 M.
30 00 B05 B11 B08. D12 I10 J08 J11 M.
13 33 B11. B11 L.
14 33 B11. C11 L.
15 00 A02. K03 O.
15 00 A12. K13 O.
15 00 A18 F10. K17 O.
16 00 A03. K02 O.
16 00 A13. K12 O.
16 00 A17. K18 O.
```

```
15 00 A05. K06 O.
34 00 A03. D09 M.
34 00 A13 E08. D14 M.
34 00 A13. E08 D13 I06 BC6 M.
15 00 A18 E10. D16 M.
13 18 B10. B10 L.
14 18 B10. C10 L.
13 18 B03. D09 M.
13 17 B02. B02 L.
35 00 .A.
36 00 .P.
37 00 .O.
50 00 .N.
```

A présent, sauvegardez le programme (sous le nom ALZAN par exemple) à l'aide du sous-programme incorporé à cet effet. L'exécution sera lancée automatiquement quand vous chargerez à nouveau le programme en mémoire. Si vous désirez au contraire une nouvelle exécution pour quelque raison que ce soit, introduisez la commande GOTO 1. Sinon la commande RUN effacerait le contenu de toutes les variables en détruisant le jeu par la même occasion.

**Et maintenant à vous de jouer...**

# COMMENT PASSER DU ZX 80 AU ZX 81

Des milliers de programmes écrits pour la version originale du ZX80 ROM 4K ne peuvent être adaptés directement sur le ZX81 (ou même sur la version ZX80 ROM 8K).

Cet appendice assez condensé vous donnera un aperçu sur la manière d'adapter un programme écrit pour la version ROM 4K du ZX80.

1- Dans le cas du ZX80 tous les indices de variable peuvent démarrer à zéro ; pour le ZX81 ils doivent obligatoirement démarrer à 1. N'importe quel programme comportant l'indice zéro doit être modifié en utilisant l'indice 1. Une méthode rapide (bien que pas toujours fiable) consiste à ajouter 1 à chaque indice que vous rencontrez dans le programme.

2- Le ZX80 utilise la fonction particulière TL$ ; celle-ci ôte le premier caractère de la chaîne afférente. Elle peut être remplacée par l'utilisation de l'expression (2 TO ) accolée à la variable alphanumérique.
Par exemple :

```
LET A$=TL$(X$)
```

devient

```
LET A$=X$(2 TO )
```

3- La fonction RND travaille d'une façon différente dans les deux cas ; en effet, pour le ZX80 ROM 4K l'argument fixe la valeur maximale du nombre aléatoire. Ainsi RND(100) donne un nombre aléatoire compris entre 1 et 100.
On peut détourner la difficulté de la façon suivante :

```
LET N=RND(X)
```

doit être à présent écrit :

```
LET N=INT (RND*X)+1
```

4- Le ZX80 ROM 4K ne manipule que les valeurs entières ; le résultat des divisions est donc systématiquement arrondi. On peut utiliser à cet effet la fonction INT dans l'expression du résultat des divisions (à moins d'avoir une bonne raison pour ne pas le faire).

5- Sur le ZX80, la virgule utilisée avec l'instruction PRINT (tabulation automatique) décale l'affichage d'un quart d'écran sur la droite (au lieu d'un demi pour le ZX81). En utilisant la fonction TAB avec précautions, il est possible de se déplacer latéralement sur la zone d'impression suivante.
Les différentes zones d'impression correspondent à :

TAB 8, TAB 16, TAB 24 et TAB 0 (début de ligne)

6- Les programmes utilisant les instructions PEEK et POKE ne peuvent être adaptés directement sur le ZX81. Nous avons déjà vu les conséquences de l'emploi de ces instructions sur le programme ; d'autre part, donner ici une liste des variables-système du ZX80 serait trop long.

7- Certaines valeurs des codes caractères ont été modifiées (particulièrement pour les caractères graphiques). Aussi, méfiez-vous des programmes contenant un grand nombre de fonctions CODE. Les valeurs des codes n'ayant pas changé sont compris entre 12 et 17, 25 et 63, 140 et 145, 153 et 191. D'autres valeurs sont également restées identiques, mais l'essentiel est de savoir que les caractères 0 à 9 et A à Z ont conservé le même code.

8- Le ZX80 exécute une boucle FOR/NEXT au moins une fois. Ceci n'est pas vrai dans le cas du ZX81, puisque la boucle est tout à fait court-circuitée si la valeur "finale" est déjà dépassée. Il est vraisemblable que cela ne vous posera pas trop de problèmes. Cependant, méfiez-vous des boucles utilisant les noms de variables en tant que valeurs de contrôle et valeurs limites.

Ces quelques différences mises à part, vous ne devriez pas rencontrer trop de problèmes ! Le meilleur moyen pour s'en assurer est de comprendre le programme d'abord puis de le ré-écrire correctement. Il a alors plus de chance de marcher convenablement.

# ZX 81 : LISTE DU SÉLECTEUR DE BLOC

Cet appendice présente la liste en assembleur du programme écrit dans le chapitre 7 ("Utilisation des codes machine").

Les adresses de tous les codes machine sont données par rapport à une adresse de départ 0000. Cette méthode permet d'implanter le programme à n'importe quel endroit de la mémoire utilisateur.

```
                    ;    ZX81 Sélecteur de bloc
                    ;
                    ;    Variables système
                    ;    -----------------
                    ;
4004               ramtop  equ    16388    ;pointeur de la dernière
                                             adresse mémoire
4007               ppc     equ    16391    ;numéro de la ligne en cours
                                             d'exécution
400C               dfile   equ    16396    ;pointeur du fichier écran
4010               vars    equ    16400    ;pointeur de la zone varia-
                                             bles
401C               stkend  equ    16412    ;pointeur de la mémoire
                                             restante
407B               spare   equ    16507    ;variable système inutilisée
                   ;
0005               maxbloc equ    5        ;nombre de blocs en sous-
                                             programmes
                   ;
0000'                      cseg
                   ;
0000'              adresdepart
                   ;      L'exécution d'une instruction d'appel à un
                   ;      sous-programme en code machine (GOSUB USR)
                   ;      provoque l'affectation de l'adresse de bran-
                   ;      chement dans le registre BC. Cette adresse
                   ;
```

```
                   ;     est utilisée comme base de calcul pour
                   ;     toutes les instructions JUMP et CALL,confi-
                   ;     nant le sous-programme à une seule page.
                   ;     La variable système RAMTOP doit contenir
                   ;     l'adresse du début d'une page mémoire
                   ;     (c'est-à-dire un multiple de 256). Ce nu-
                   ;     méro de page est affecté au registre H et
                   ;     doit rester identique pendant l'exécution
                   ;     du sous-programme.
                   ;     Le numéro d'appel du bloc de sous-programme
                   ;     est placé dans l'octet le moins significa-
                   ;     tif de la variable système "SPARE".

0000'   3A 407B          ld    a,(spare)   ;charge le numéro du bloc
0003'   FE 05            cp    maxbloc     ;numéro de bloc correct ?
0005'   D0               ret   nc          ;retourne si incorrect
0006'   5F               ld    e,a         ;copie l'accumulateur dans
                                            le registre pair DE
0007'   16 00            ld    d,0
0009'   21 0010'         ld    hl,tablebloc   ; adresse des blocs
000C'   60               ld    h,b         ;copie le numéro de page
                                            dans H
000D'   19               add   hl,de       ;additionne le numéro de
                                            bloc
000E'   6E               ld    l,(hl)      ;obtient l'adresse du
                                            sous-programme
000F'   E9               jp    (hl)        ;va à l'adresse du sous-
                                            programme

0010'              tablebloc
                   ;     L'introduction de la valeur zéro dans
                   ;     "SPARE" provoque un saut à la première
                   ;     adresse de cette table. Chaque octet
                   ;     entré donne la valeur relative du dépla-
                   ;     cement à effectuer par rapport à l'adresse
                   ;     départ pour obtenir l'adresse du bloc dési-
                   ;     ré.

0010'   15               defb  (fonc0-adresdepart) and 0ffh
0011'   22               defb  (fonc1-adresdepart) and 0ffh
0012'   27               defb  (fonc2-adresdepart) and 0ffh
0013'   2C               defb  (fonc3-adresdepart) and 0ffh
0014'   31               defb  (fonc4-adresdepart) and 0ffh

0015'              fonc0:
                   ;     Donne l'estimation de la mémoire disponible
0015'   21 0000          ld    hl,0        ;efface le contenu de HL
0018'   39               add   hl,sp       ;obtient le pointeur de
                                            pile
```

```
0019'   ED 5B 401C        ld      dl,(stkend);fin de la mémoire système
001D'   ED 52             sbc     hl,de       ;obtient la différence
001F'   44                ld      b,h         ;met la réponse dans BC
0020'   4D                ld      c,l
0021'   C9                ret                 ;retourne  au Basic

0022'             fonc1:
                  ;       Donne l'adresse du fichier écran
0022'   ED 4B 400C        ld      bc,(dfile) ;copie le contenu  de la
                                              variable système
0026'   C9                ret                 ;retourne au Basic

0027'             fonc2:
                  ;       Donne l'adresse des variables du Basic
0027'   ED 4B 4010        ld      bc,(vars)  ;copie le contenu de VARS
002B'   C9                ret

002C'             fonc3:
                  ;       Donne l'adresse de la mémoire libre
                  ;    (au-delà de ce sous-programme)
002C'   01 0036           ld      bc,limiteprogram-adresdepart
002F'   44                ld      b,h
0030'   C9                ret

0031'             fonc4:
                  ;       Donne le numéro de ligne en cours
0031'   ED 4B 4007        ld      bc,(ppc)   ;copie le contenu de la
                                              variable système PPC

0036'             limiteprogramequ      $  ;occupation de la mémoire

                          end
```

# SOLUTIONS DES PROBLÈMES

## TEMPS D'EXECUTION DES PROGRAMMES DE REFERENCE

Voici les resultats obtenus sur le ZX81 en mode lent. Les temps ont été déterminés à l'aide d'un chronomètre digital d'une valeur de 100 Francs environ.

| Nom | Temps d'exécution |
|---|---|
| PR1A | 24.9 |
| PR1B | 25.4 |
| PR2A | 24.9 |
| PR2B | 35.4 |
| PR3A | 24.9 |
| PR3B | 37.5 |
| PR4A | 24.9 |
| PR4B | 61.8 |
| PR5A | 31.9 |
| PR5B | 26.8 |
| PR6A | 22.8 |
| PR6B | 25.5 |
| PR7A | 27.0 |
| PR7B | 17.8 |

Un dernier point. Dans le jeu des questions et des réponses (page 14 ), une question vous avait été posée, à savoir : comment répondre à une question correctement sans même la lire ?

C'est très simple. A chaque fois qu'une question est posée, la réponse est déjà affectée à W$. En effaçant les guillemets (EDIT) et en entrant W$ comme solution, vous aurez immanquablement la réponse correcte ! De toute façon, qui aime les questions... ?

# INDEX DES MOTS BASIC

| Nom | Signification |
|---|---|
| ABS(n) | donne la valeur absolue de n |
| ACS(n) | valeur en radians de Arc Cosinus n |
| AND | opérateur logique |
| ASN(n) | valeur en radians de Arc Sinus n |
| AT y,x | commande l'affichage à la ligne y colonne x |
| ATN(n) | Arc Tangente n en radians |
| CHR$(n) | donne le caractère correspondant au code n |
| CLEAR | efface toutes les variables |
| CLS | efface l'écran et positionne l'affichage en haut à gauche |
| CODE(s) | donne le code du premier caractère de la chaîne s |
| CONT | reprend l'exécution du programme après |
| COPY | copie l'affichage écran sur l'imprimante |
| COS(n) | donne la valeur en radians de Cosinus n |
| DIM | dimensionne un tableau (alphabétique ou numérique) |
| EXP(n) | fournit la valeur de $e^n$ (exponentielle) |
| FAST | met le ZX81 en mode rapide |
| FOR | introduit une variable de contrôle dans une boucle |
| GOSUB n | branche au sous-programme de la ligne n |
| GOTO n | branche le programme à la ligne n |
| IF e THEN | clause THEN réalisée si l'expression e est vrai |
| INKEY$ | fournit le caractère correspondant à la touche (s'il y a lieu) |
| INPUT | entrée d'expressions numériques ou alphabétiques au clavier |
| INT(n) | arrondit n à la valeur entière par défaut |
| LEN(s) | donne la longueur de la chaîne afférente |
| LET | affectation des variables |
| LIST | affiche la liste du programme |
| LLIST | écrit la liste du programme sur l'imprimante |
| LN(n) | fournit la valeur de $\log_e(n)$ (logarithme népérien) |
| LOAD | charge un programme donné à partir d'une cassette |
| LPRINT | écrit les expressions afférentes sur l'imprimante |

| | |
|---|---|
| NEW | efface totalement le contenu de la mémoire du ZX81 |
| NEXT | actualise la variable de contrôle à la valeur suivante |
| NOT(n) | inverse le résultat de la comparaison vraie de l'expression n |
| OR | opérateur logique |
| PAUSE | suspend l'exécution pendant n/50 secondes |
| PEEK(n) | donne la valeur de l'octet situé à l'adresse n |
| PI | donne la valeur 3.1415926 |
| PLOT | affiche en noir l'élément d'image de coordonnées x,y |
| PRINT | place les expressions afférentes dans le fichier d'affichage |
| RAND | affectation de la base aléatoire |
| REM | permet des annotations au début d'un programme |
| RETURN | termine un sous-programme |
| RND | fournit la valeur du nombre aléatoire 0<=n<1 |
| RUN | efface le contenu des variables et lance l'exécution du programme |
| SAVE | sauvegarde un programme donné sur cassette |
| SCROLL | fait sauter une ligne à l'affichage |
| SGN(n) | fournit le "signe" de n (-1,0 ou +1) |
| SIN(n) | donne la valeur en radians de Sinus n |
| SLOW | place le ZX81 en mode lent (mode calcul et affichage) |
| SQR(n) | donne la racine carrée de n |
| STEP n | incrément n effectué au cours d'une boucle |
| STOP | interrompt l'exécution du programme en donnant le code erreur 9 |
| STR$(n) | donne la correspondance alphabétique du nombre n |
| TAB(n) | déplace l'affichage à la colonne n |
| TAN(n) | donne la valeur en radians de Tangente n |
| UNPLOT | efface l'élément d'image de coordonnées x,y |
| USR(n) | appelle le sous-programme en codes machine situé à l'adresse n de la mémoire |
| VAL(s) | donne la valeur numérique de la chaîne s (si cela est possible) |

***Renseignements utiles*** :

**CODE** "Ø" - 28
**CODE** "A" - 38

| | | | |
|---|---|---|---|
| FRAMES | 16436/7 | VARS | 16400/1 |
| RAMTOP | 16388/9 | DFILE | 16396/7 |

Le programme démarre à 16509

# CODE ERREUR

| Code erreur | Signification |
|---|---|
| 0 | |
| 1 | la variable de contrôle n'existe pas |
| 2 | nom de variable non défini |
| 3 | indice en dehors des limites |
| 4 | mémoire insuffisante |
| 5 | fichier d'affichage écran complet |
| 6 | dépassement de capacité arithmétique |
| 7 | instruction RETURN trouvée sans GOSUB |
| 8 | INPUT utilisé en mode direct |
| 9 | commande STOP exécutée |
| A | fonction numérique incorrecte (exemple :SQR -1) |
| B | valeur entière en dehors des limites (PRINT, AT,PLOT, etc.) |
| C | l'argument de VAL n'est pas une expression numérique |
| D | interruption du programme (BREAK ou STOP dans INPUT) |
| E | inutilisé |
| F | absence de nom de la commande SAVE |

Achevé d'imprimer en Janvier 1982
sur les presses de l'imprimerie O.M
77330 OZOIR LA FERRIERE
Dépôt légal : Janvier 1982
N° d'impression : 375
N° d'édition : 86595-36-1
ISBN : 2-86595-036-0

ISBN 2.86595.036.0
Imprimé en France

P.S.I.

Editions du P.S.I. 41-51, rue Jacquard BP 86 77400 Lagny-s/Marne Téléphone (6) 007.59.31

IMP. OM · 77330 OZOIR-LA-FERRIERE ·


LE PETIT LIVRE DU ZX81
TOMS T.
P.S.I.

# AMSTRAD CPC

## MÉMOIRE ÉCRITE
## MEMORY ENGRAVED
## MEMORIA ESCRITA




https://acpc.me/